CSCI 566: Deep Learning and its Applications

Yue Zhao
Thomas Lord Department of Computer Science
University of Southern California

Credits to previous versions of USC CSCI566, CMU 10601/701, Stanford CS 229, 231n

Logistics

Project teams:

- 4-6 people – talk to me if you have fewer/more people
- Use the break and after-class time to form a team ☺
- We have a signup sheet
    - We will assign one for you if you face difficulty forming a group by the end of the week (we will have to match people to team)
    - Please sign up the google sheet to be assigned a group:
    - https://docs.google.com/spreadsheets/d/1IEisSea4CF86ikrhrUKtiTDqQQ-SkctIWU5XTHu8cr8/edit?usp=sharing

Logistics

Project teams:

- 4-6 people – talk to me if you have fewer/more people
- Use the break and after-class time to form a team ☺

https://docs.google.com/spreadsheets/d/1IEisSea4CF86ikrhrUKtiTDqQQ-SkctIWU5XTHu8cr8/edit?usp=sharing

TAs have also posted a few ideas on piazza – also a good way to form a group (by topics). But maybe it is a bit too late…

Logistics

Project teams:

Project Name/Title (Temporary)	Member 1	Email	Member 2	Email	Member 3	Email	Member 4
Using CNN to identify cute cat (Example)	yue	yue@usc.edu	yue	yue@usc.edu	yue	yue@usc.edu	
DeepFake Video Detection	Aiden Chang	aidencha@usc.edu	Adrian Roman	romanguz@usc.edu	Hyunkeun Park	hyunkeup@usc.edu	Kevin Hopkins
Offline RL with learned action discretization with VQ-VAE	Jonathan Zamora	jz_605@usc.edu	Charlene Yuen	yuenchar@usc.edu	Jung Whan Lee	jlee7870@usc.edu	Mann Patel
Something NLP related/Flexible	Javin Liu	javinliu@usc.edu	Dennis Perepecl	perepech@usc.edu	Aaron Ahmed	arahmed@usc.edu	
LLMath - A neuro-symbolic approach to Math in LLMs	Akhil Krishna Red	akhilkri@usc.edu	Ashmika Ajay G	ashmikaa@usc.edu	Harsh Thakkar	harshtha@usc.edu	Rishabh Budhia
Reinforcement Learning for Recommender Systems: 1. Environment Suite	Kashish Madan	kmadan@usc.edu	Gurpreet Kaur S	saimy@usc.edu	Pranav Parnerka	parnerka@usc.edu	Dhanush Lingegov
Email Spam Detection	Vyas Ram Srinivas	vrsriniv@usc.edu					

Existing Teams | Looking for Team

Logistics

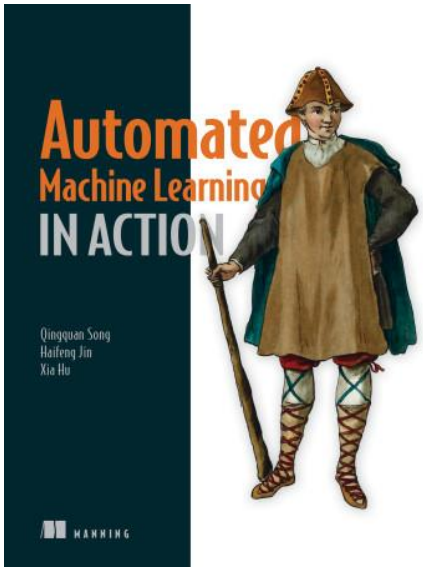
Preproposal:

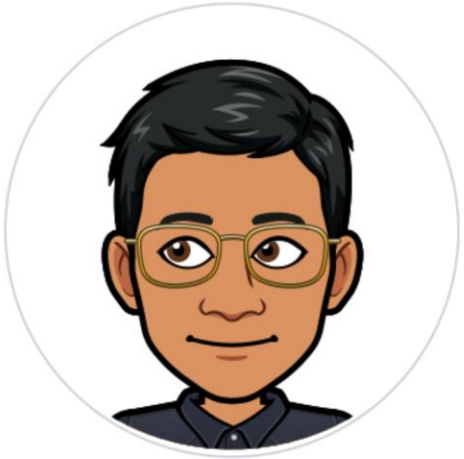
- Due on Feb 2nd 11:59 pm (Next Friday)
- Submit via Gradescope
- Team submission (one submission per group – link in the system)
- 1-2 pages, PDF format
    - why is it useful/interesting
    - what is your potential solution for it,
    - expected outcomes,
    - and a rough timeline.

Invited Talk 1: Dr. Haifeng Jin

Guest lecture on March 1st

Haifeng Jin is a software engineer on the Keras team at Google. He is the creator of **AutoKeras**, coauthor of **Keras Tuner**, and a contributor to **Keras** and **TensorFlow**. Haifeng got his Ph.D. in computer science at Texas A&M University. His research interest is automated machine learning (AutoML).





Haifeng Jin

haifeng-jin · he/him

Unfollow

Software engineer on Keras | Project lead of AutoKeras & KerasTuner

690 followers · 143 following

Followed by yifanjiang19

@keras-team

https://haifengjin.com/about

Pinned

keras-team/autokeras Public

AutoML library for deep learning

Python 9k 1.4k

keras-team/keras-tuner Public

A Hyperparameter Tuning Library for Keras

Python 2.8k 385

datamllab/automl-in-action-notebooks Public

Jupyter notebooks for the code samples of the book "Automated Machine Learning in Action"

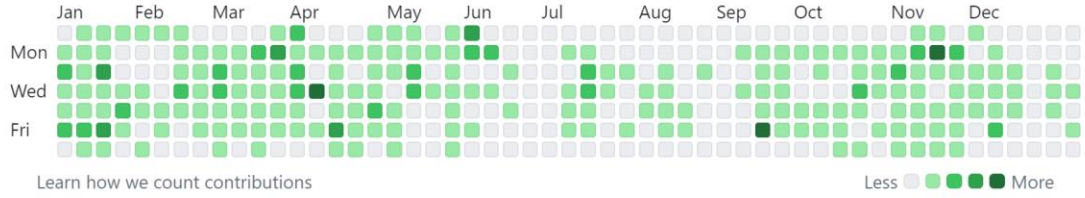
Jupyter Notebook 77 37

keras-team/keras Public

Deep Learning for humans

Python 60.2k 19.5k

956 contributions in the last year



Learn how we count contributions

Less More

2024

2023

2022

2021

2020

2019

2018

2017

@keras-team @tensorflow @conda-forge More

Activity overview

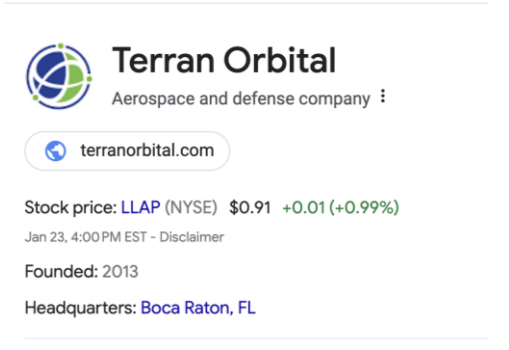
Contributed to keras-team/keras-tuner, keras-team/keras, keras-team/keras-core

15%
Code review

Invited Talk 2: Valentino Constantinou

Guest lecture on March 22th

Valentino Constantinou is an experienced leader in analytics and artificial intelligence for the **aerospace** industry, serving as a Senior Data Scientist and Team Lead at Terran Orbital.



At NASA JPL operated by CalTech, he served as the Principal Investigator to a multi-year alarm analytics effort, co-organized a monthly meetup of the Lab's open source developers (the Open Developer Meetup).

Valentino Constantinou

vc1492a

Follow | Sponsor

Senior Data Scientist @ Terran Orbital
Open Source Software Enthusiast

167 followers · **216** following

- Terran Orbital
- Pasadena, California
- 22:56 - same time
- https://www.valentino.io/
- https://scholar.google.com/citations?

Pinned

PyNomaly Public

Anomaly detection using LoOP: Local Outlier Probabilities, a local density based outlier detection method providing an outlier score in the range of [0,1].

Python ☆ 296 ⑂ 36

Hey-Waldo Public

Labeled images of the Where's Waldo puzzle for use in classification and image recognition problems.

Python ☆ 166 ⑂ 35

Raspberry-Pi-Fan-Control Public

Simple Python and shell scripts to run a fan on a Raspberry Pi using CPU temperature.

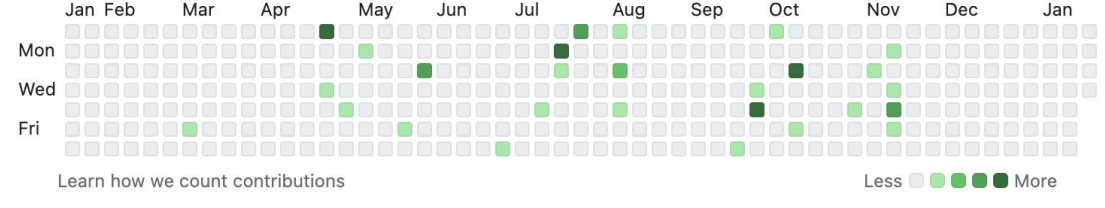
Python ☆ 44 ⑂ 7

mieda Public

Merging of set-containing Intervals Efficiently with a Directed-graph Algorithm

Jupyter Notebook ☆ 6 ⑂ 1

62 contributions in the last year



2024
2023
2022
2021
2020
2019

Activity overview

2%
Code review

Invited Talk 3: Howie Xu

Guest lecture on ~~March 15th~~ (TBA)

Investor
Reality Defender
Dec 2023 - Present · 2 mos
United States

Enterprise-grade Deepfake detection startup



Senior Vice President, Engineering and AI/ML
Palo Alto Networks · Full-time
2023 - Present · 1 yr 1 mo
United States · On-site

- Lead core security products, AIOps, Autonomous Digital Experience Management (ADEM), and Cortex Data Lake... ...see more


Invited Talk 3: Howie Xu

Guest lecture on ~~March 15th~~ (TBA)

Post

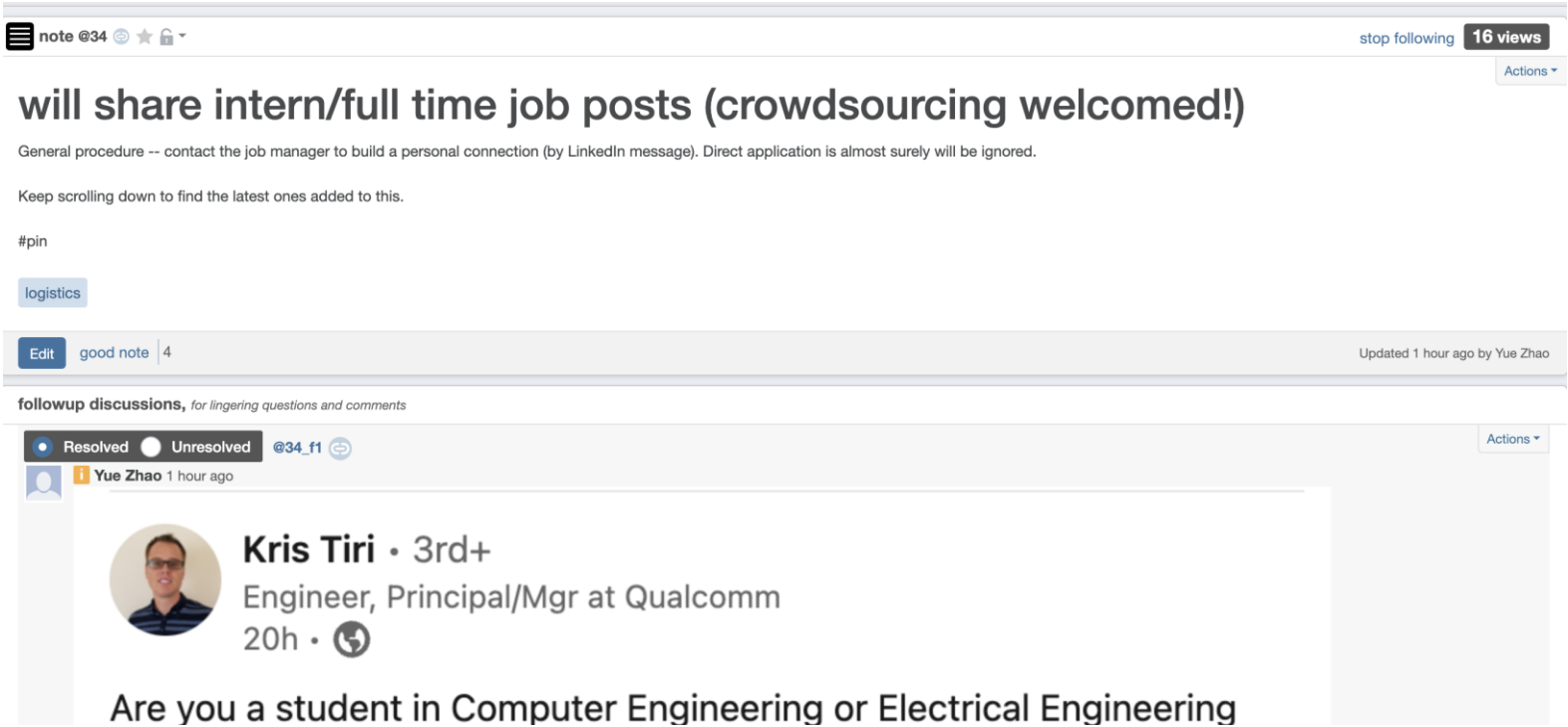
My conversation with Jerry Liu, CEO/Founder of the well-known open-...

Generative AI in Enterprise Builder's Insights | Supercloud 5

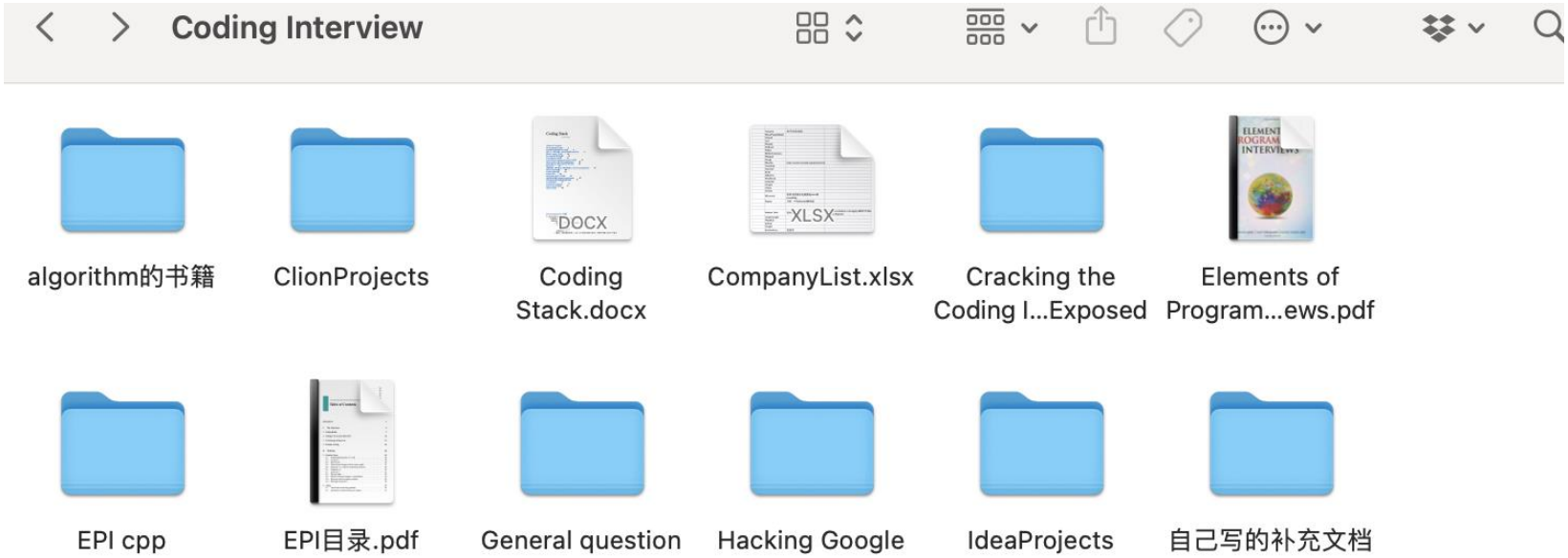
youtube.com

160 · 1 comment

Additional Things – Job Posts



Additional Things – Wishing the Best



- 优化的答案和 5.1 一样，那就是 build 一个 lookup table 出来。用 bitmask 等方法进行 shift。此处 build lookup table 就可以用上一问介绍的方法，用中间对劈开做 for loop。

5.7 Compute x^y(求多少次方) NP27， 同时 5.5 和 5.6 分别讨论了做乘法和除法

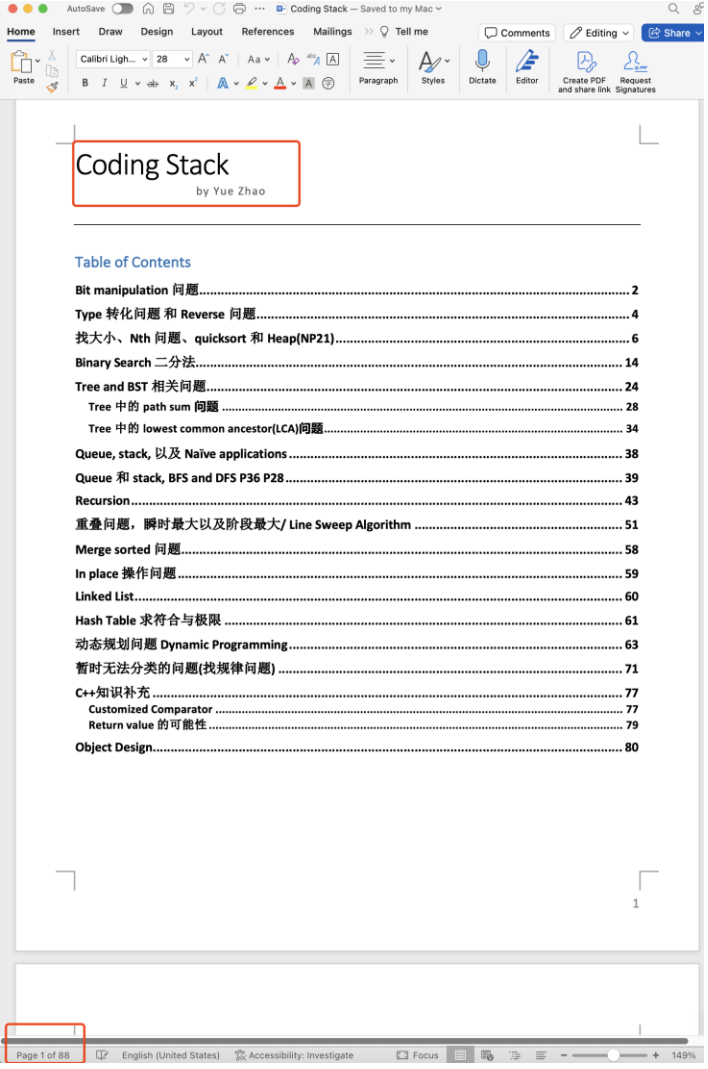
- 这题没有太多要注意的，就是注意 check Neg 和移动 bit 要小心别出错。
- EPI 说与其让 x 乘以 y 次还不如，每次 x*=x，并使 y 减半。

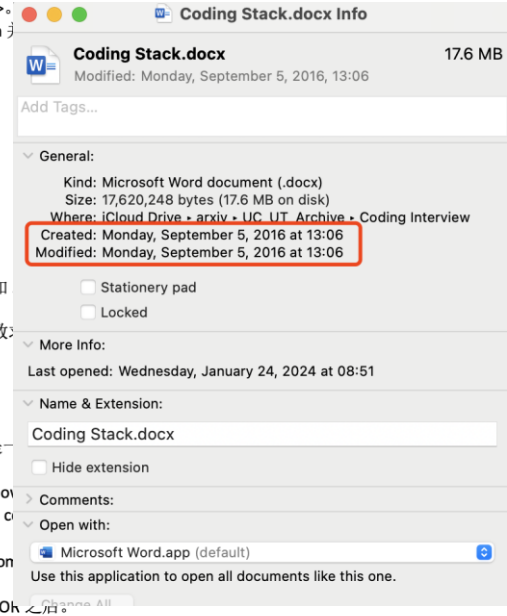
12.12 Find missing and duplicates NP19

- 第一种情况，就是只是 missing 一个 value。比如<0,1,2,3,4,5> 中只有 <0,1,2,3,5>。
  - 解法 1：因为知道 miss 了一个数，且唯一。只需要求两个 vector 的 sum 并减。
    - 效率是 O(n) 因为相加 missing array 只能 one by one。
    - 另一个问题是很容易 overflow，如果数字特别多且特别大。
  - 解法 2：利用 bitwise XOR

A	B	Q
0	0	0
0	1	1
1	0	1
1	1	0

    - XOR truth table
    - 通过这个可以看出如果两个数相同，那么起 XOR 结果就是 0. 比如 == 0
      因此只要把完整 array 里面的每个数都和 missing array 中的每个数 可。剩下的就是 missing 的那一个数。
    - 通过这个方法避免了 overflow。Complexity 不变。
- 第二种情况，多了一个数。比如<0,1,2,3,4,5> 和 <0,1,2,3,4,4,5>
  - 在这种情况下，上面的解法 1 和解法 2 依然成立。
- 第三种情况，duplicate 一个数的同时，miss 了一个数。这样 vector 的长度还是一 如<0,1,2,3,4,5> 和 <0,1,3,4,4,5> 中 2 miss 了 4 重复了。
  - 解法 1：可以算 sum 和 product，得到两个式子来求解。但太容易 overflo
  - 解法 2：直接 sort 在一个个的 traversal。O（nlogn）因为 sorting，space c 是 O（1）.
  - 解法 3：用 hash table 储存每个 key 和次数。在 traversal check。Space con 和 time complexity 都是 O（n）。
  - 解法 4：依然利用 XOR 的特性，将完整的 vector 和错误的 vector 进行 XOR 之后。得到的数字就是 （missing）XOR（duplicate）的结果：
    - 在这道题里面得到的是 2^4 = 110，110 的含义就是 missing 和 duplicate 的数字的 bit 2， 和 bit 1 不相同，但 bit 0 是相同的。<0,1,2,3,4,5> XOR <0,1,3,4,4,5>就等于 2^4 = 110。
    - 因此如果令所有（完整的 vector 和错误的 vector 中）bit 1 == 1 的数一起求 XOR，那么至少我们可以得到 missing 或者 duplicate 中的以后。





USC Viterbi School of Engineering

Quizzes

Week 4 Feb 2	Neural Network Basics - Perceptron Revisited - Gradient Descent - Forward Propagation Deep Learning Software Tutorial	Quiz 1	Course Project Teams Formed; Pre-proposal DUE Google Cloud Platform - 2024.pdf

Feb 2nd, next lecture in class – at the beginning

- If you could not make it for valid reason, let me know

Quizzes are graded based on completion – so do not worry too much about the correctness.

- Multiple choices

- No mathematical proof or other derivatives

Neural Networks

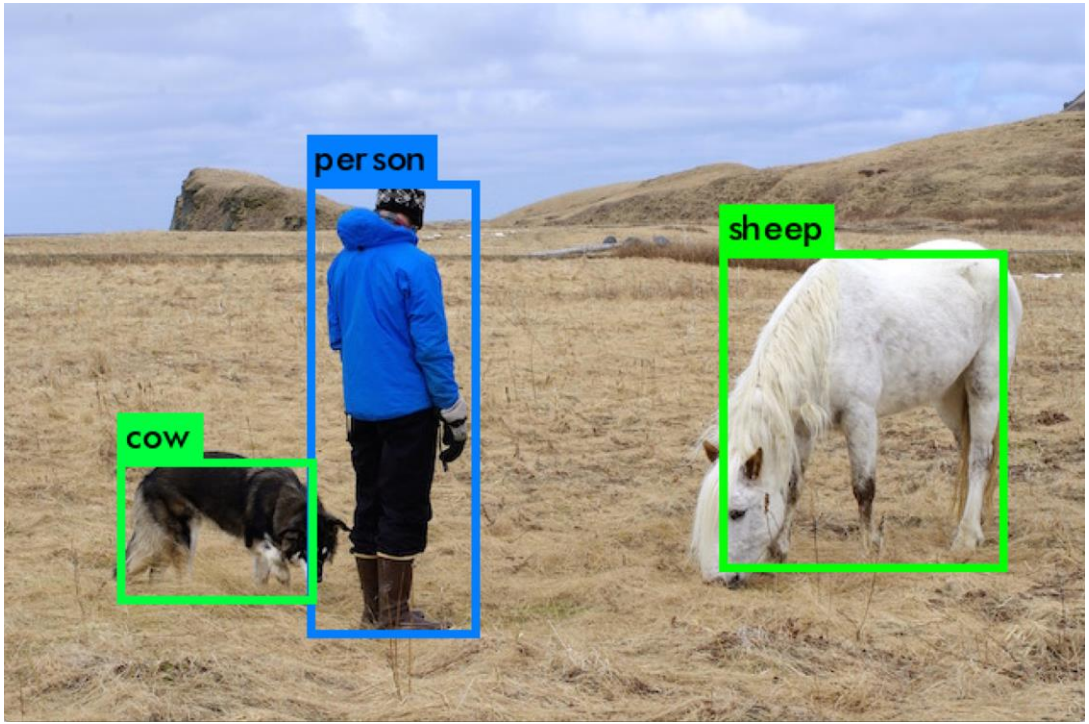
Basics

It’s a matter of One Function

Really…?

Let’s take a look

Object Detection

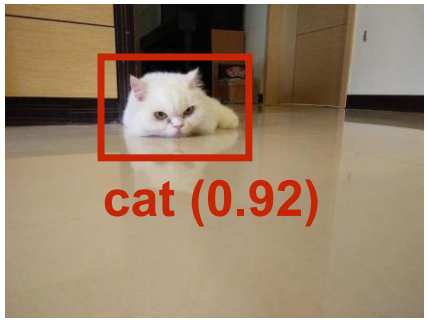


Object Detection

X:
Image

f(X)

Y:
Object label
&
Bounding box



Types of Learning

- Supervised Learning — desired output (**Y**) in training data
- Unsupervised Learning — **Y** not in training data
- Weakly / Semi-supervised Learning — some of **Y** in training data
- Reinforcement Learning — rewards based on a set of actions

Our goal is “to approximate”

There may exist an exact function (f*) mapping from X to Y.

Our goal is not to find this exact function.

Rather, we are happy as long as f(X) can **approximate** f*(x). f does NOT have to be exactly f*.

Our goal is “to approximate”

For f(X) to approximate any f*(X), f is better to be highly capable.

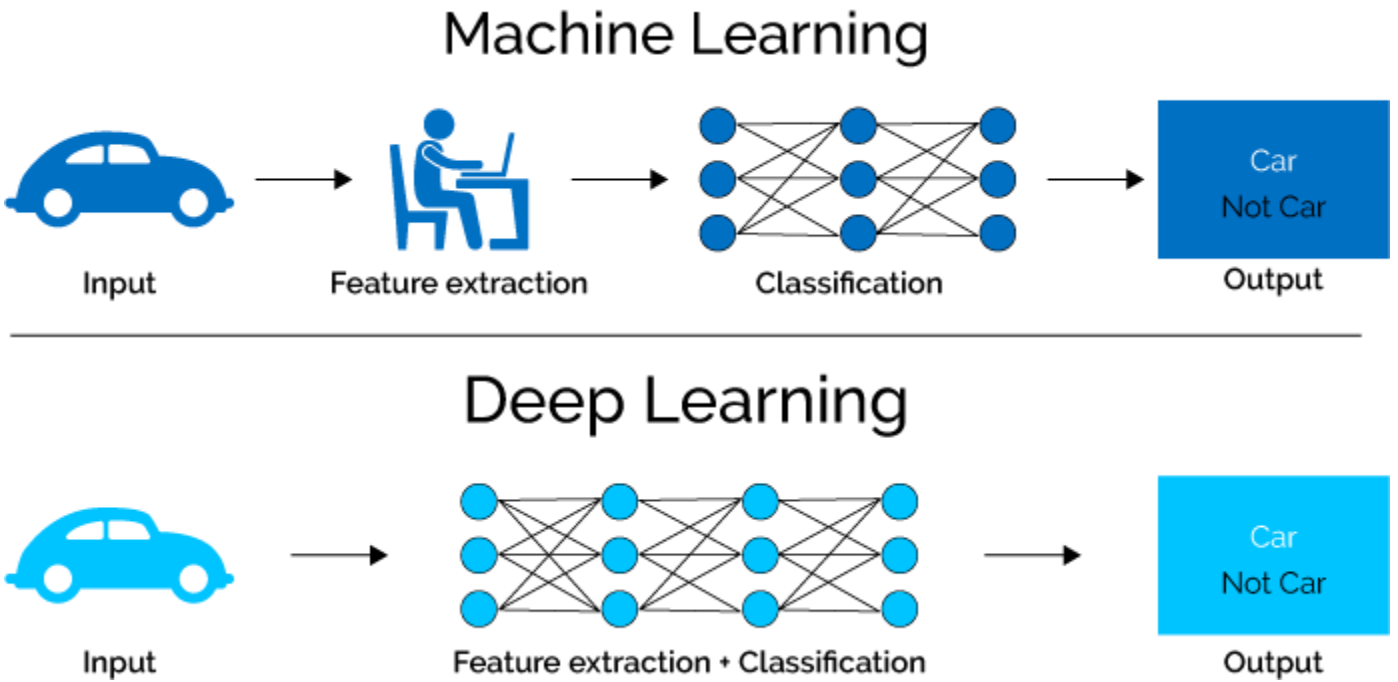
Deep learning is an effective method for this

- Non-linear (high capacity)
- Hierarchical
- End-to-End learning

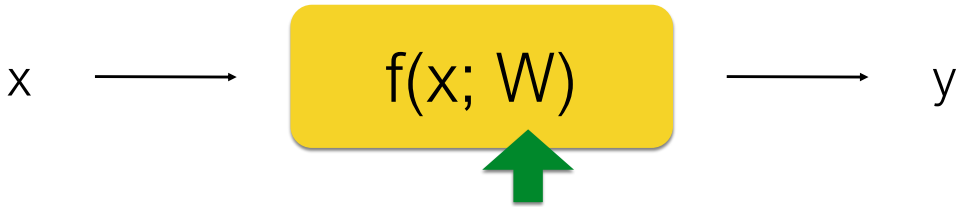
Recap: Our course

(1) How do we learn this function (using ML and DL)?
(2) How to formulate a problem into this?

Recap: Classical ML vs. DL (Better Representations)



Loss function & Optimization



- **Loss function (L)** measures how well learned W can map X to Y (compared to y*).
- **Optimization** finds the best W given a loss function L (i.e. finding W that minimizes L).

Optimization

How do we find W minimizing L?

* Random search
* Analytic solution
* Numerical approach (gradient descent)

L(W; f,x,y*)

As an example, let's pick a simple loss function.

$$L(W) = || f(x; W) - y^* ||_2$$

Numerical Solution (gradient descent)

x

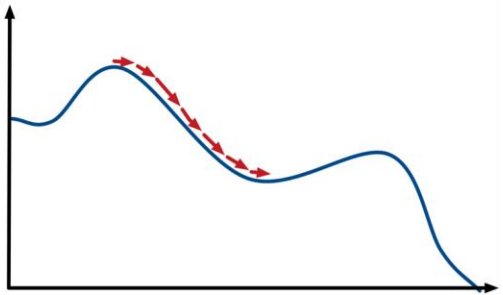
?	?	?	?
?	?	?	?
?	?	?	?

W

0
1
0

y* (ideal output)

$L(W) = || f(x; W) - y^* ||_2$



The **derivative** of a function in one-dimension

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

Numerical Solution (gradient descent)

-0.1	0.2	0.15	0.05
0.25	0.05	0.2	-0.1
0.25	0.15	-0.15	0.4

W

-0.1+ 0.001	**0.2+ 0.001**	0.15	0.05
0.25	0.05	0.2	-0.1
0.25	0.15	-0.15	0.4

W+h

0.5	**0.9**	?	?
?	?	?	?
?	?	?	?

gradient (dW)

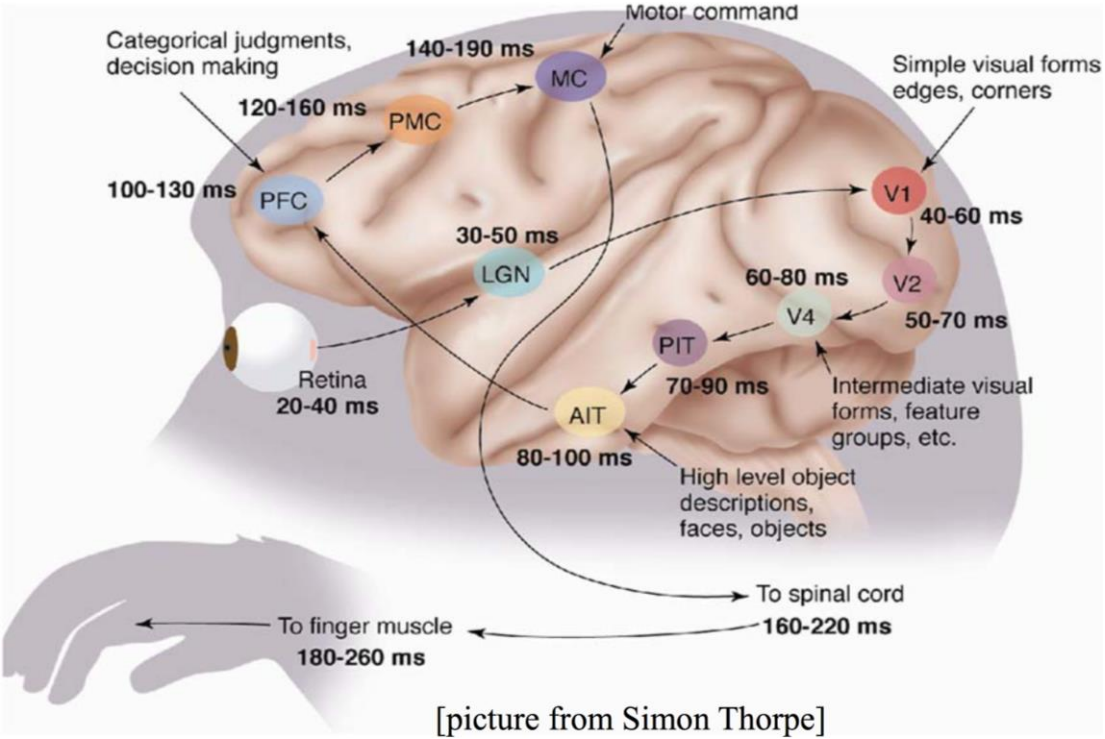
L(W) = 1.6688

L(W+h) = 1.6697

Gradient = (1.6697-1.6688)/0.001 = 0.9

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

Deep Learning may be Motivated by Human Brain



[picture from Simon Thorpe]

Be Careful with Brain Analogies!

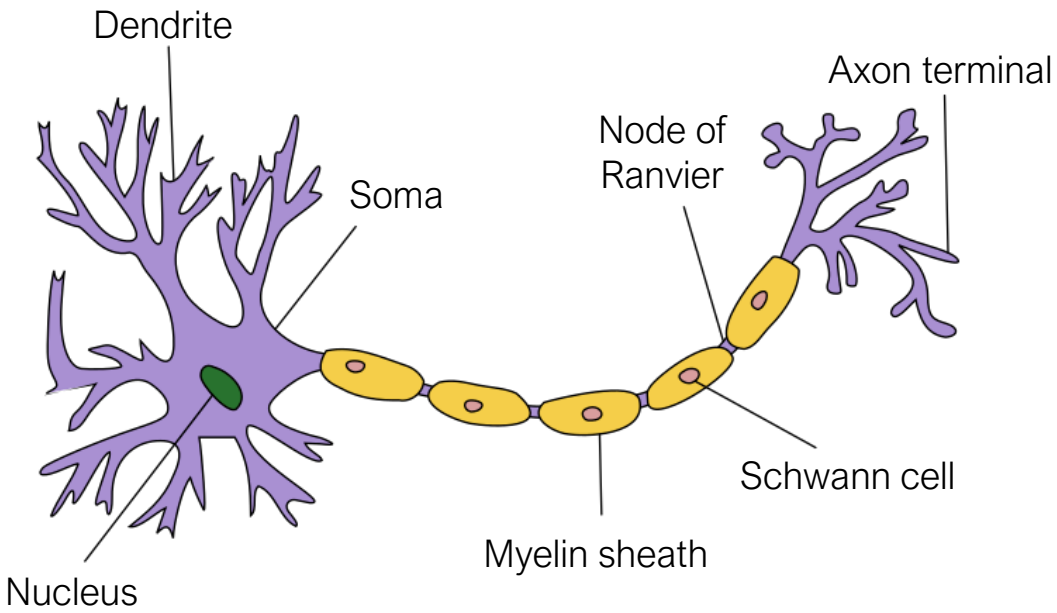
Biological Neurons:

- Many different types
- Dendrites can perform complex non-linear computations
- Synapses are not a single weight but a complex non-linear dynamical system

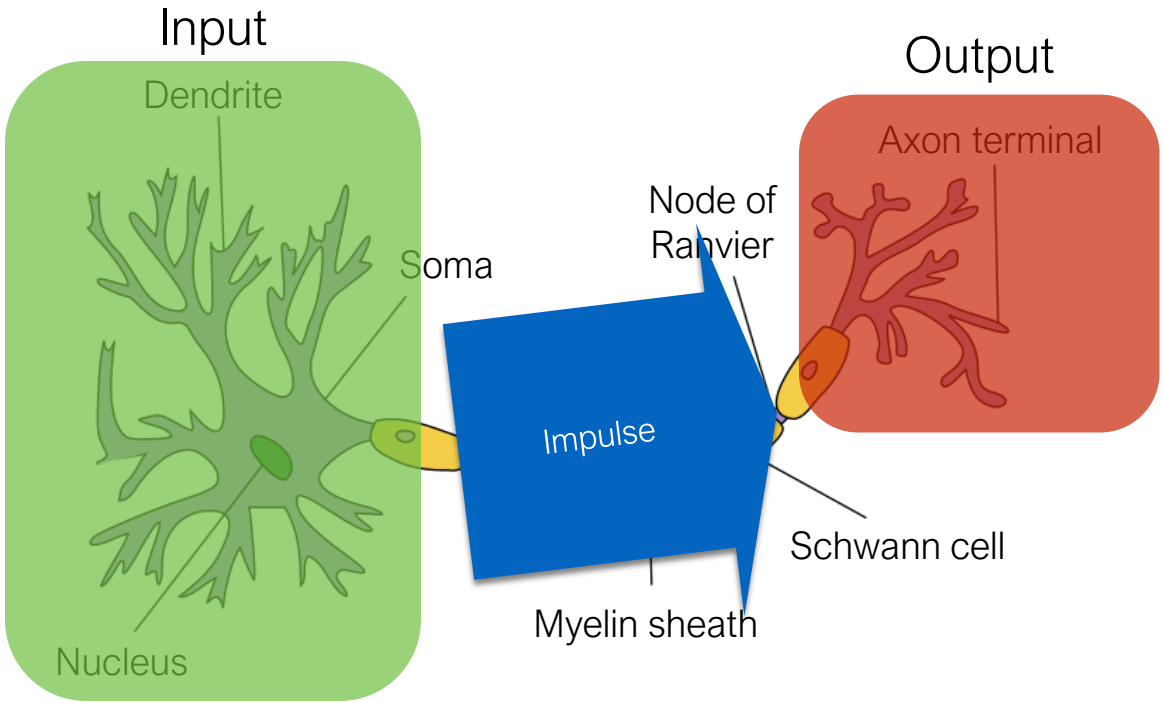
These are good motivations, while the actual relationship is weak.

[Dendritic Computation. London and Hausser]

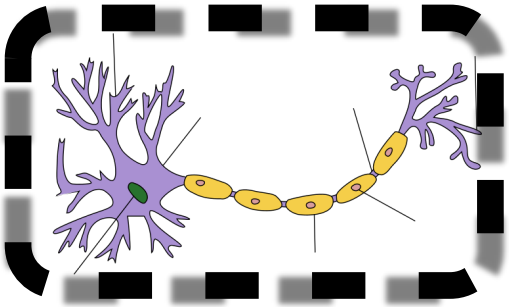
Neuron


Dendrite
Axon terminal
Node of
Ranvier
Soma
Schwann cell
Myelin sheath
Nucleus

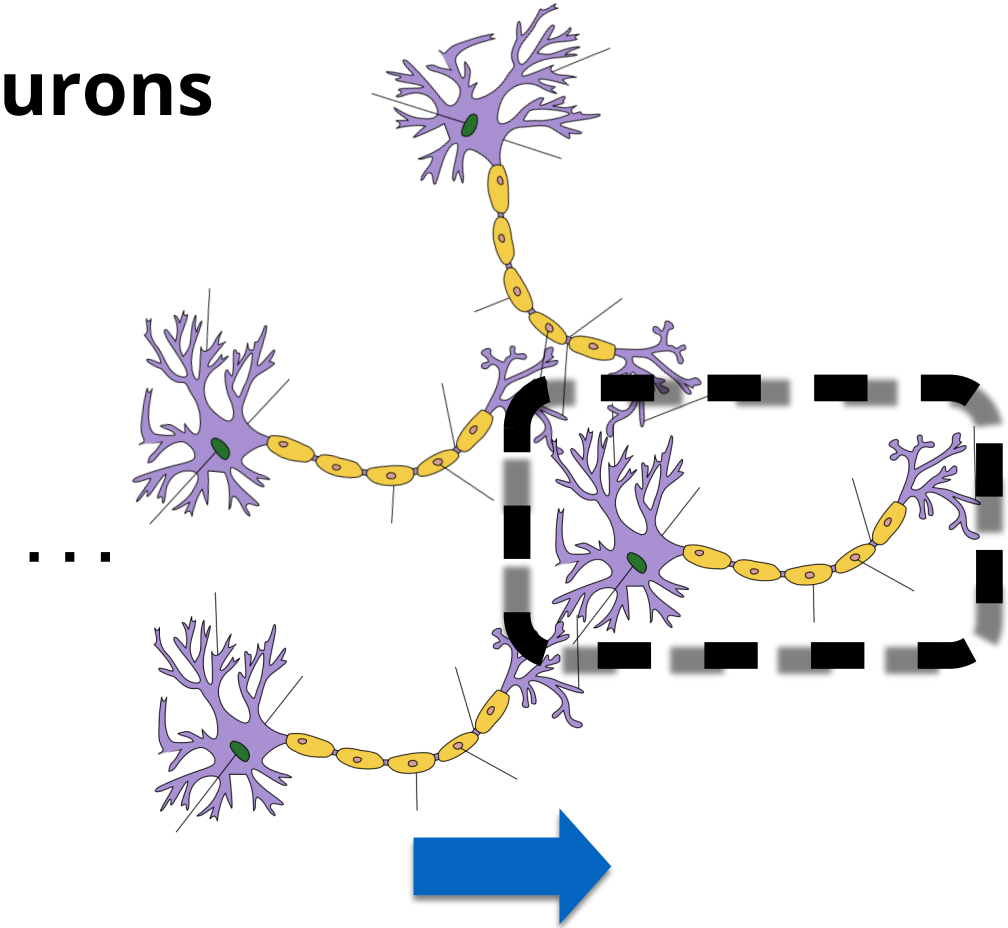
Neuron



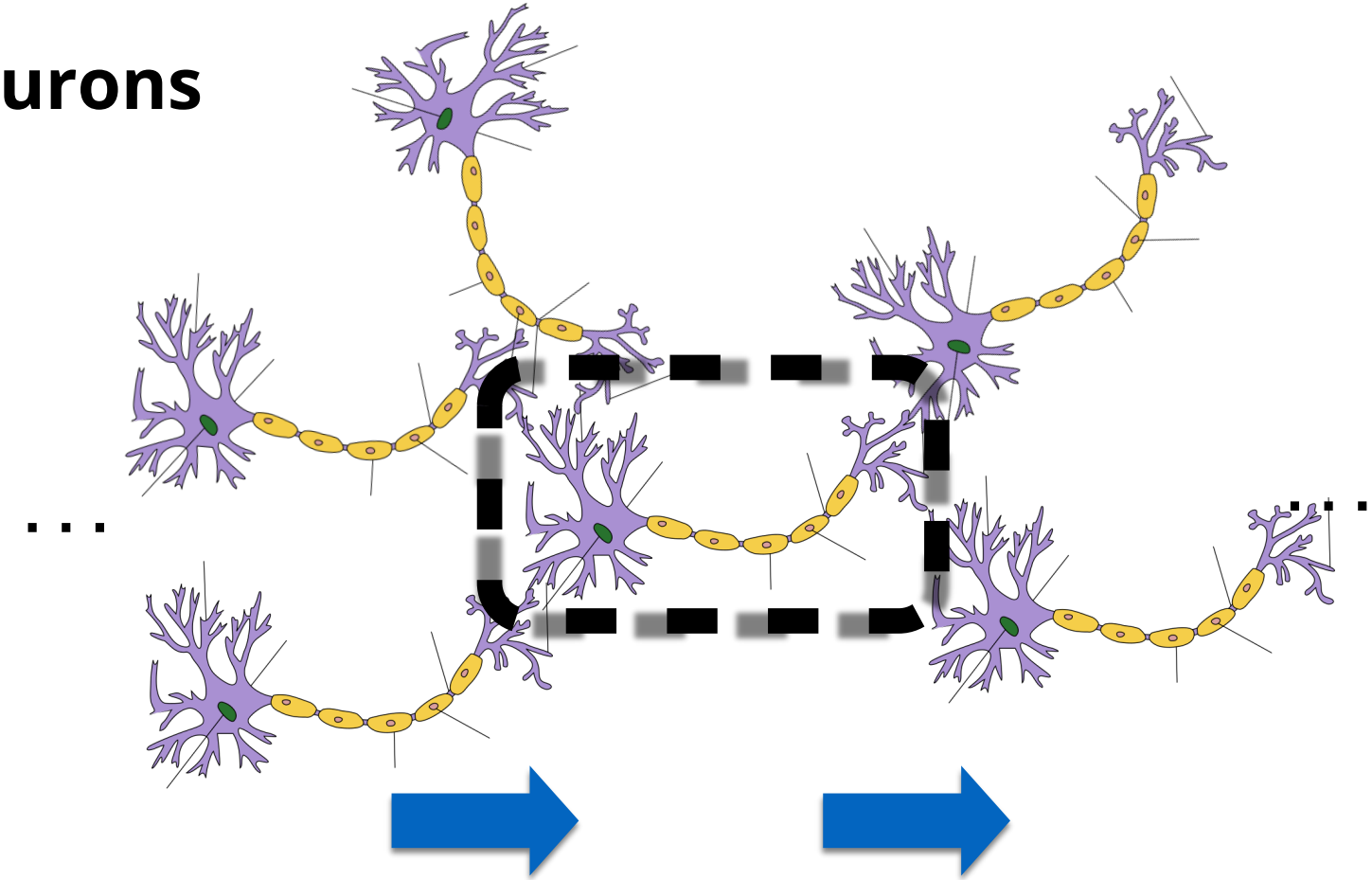
Neuron



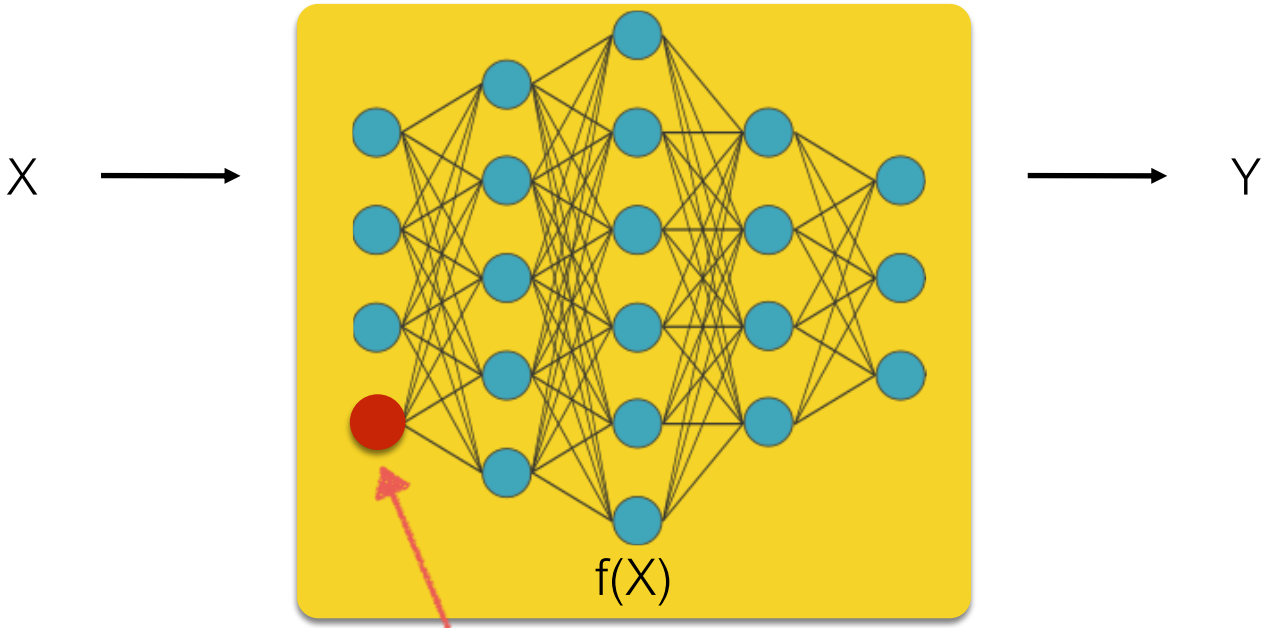
Neurons



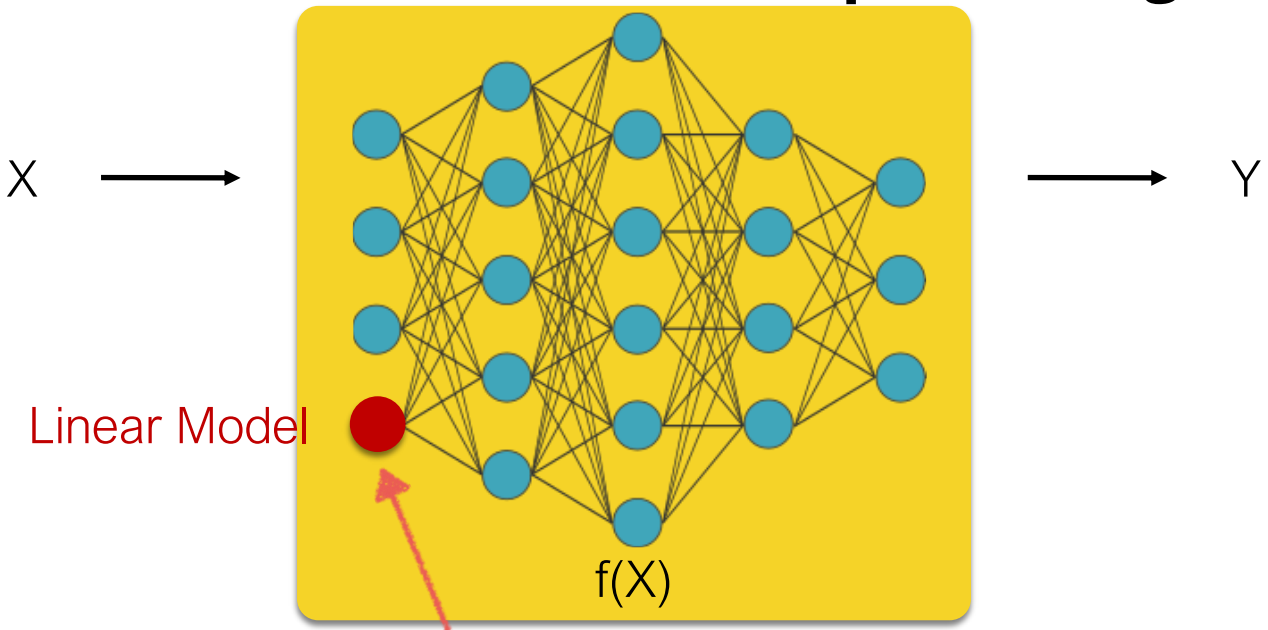
Neurons



From Neuron to Neural Networks



From Linear Classification to Deep Learning



Apparently, this is a unit used in deep neural networks too.

Image classification by Linear Classification

5x1 5x7500 7500x1

$f(x; W) = Wx + b$ 5x1

X → f(X; W) → Y

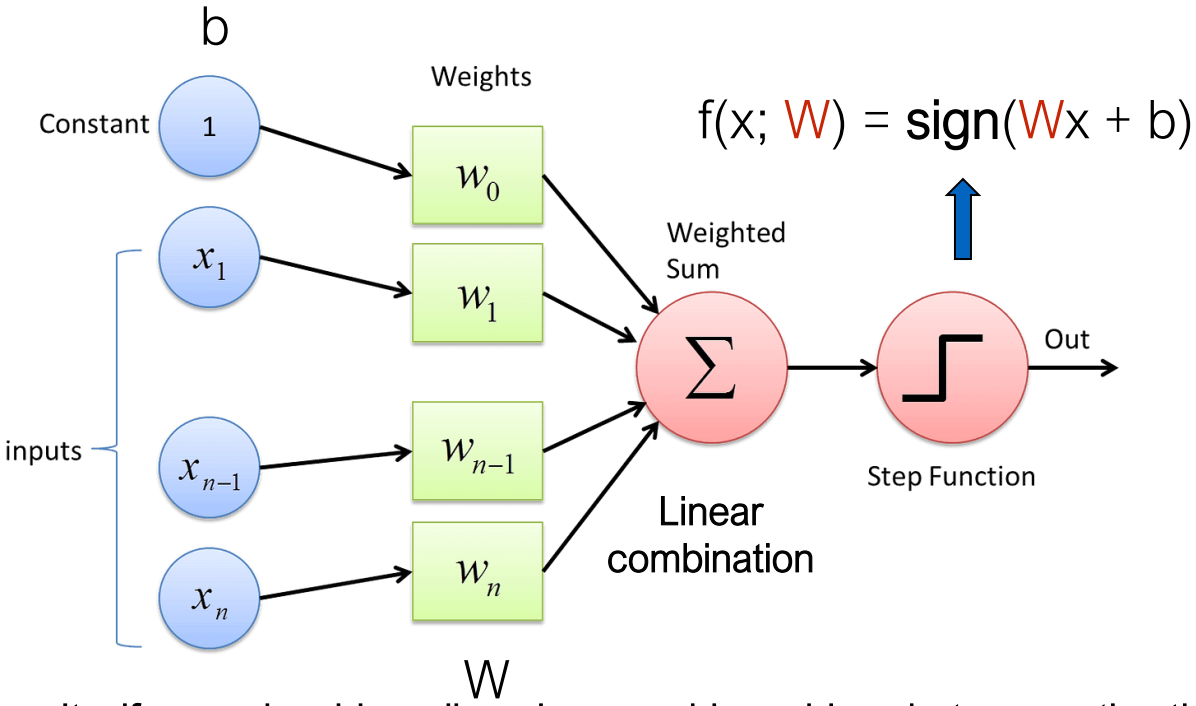
50x50x3 numbers
(7500 numbers)

Weights
(that we want to learn)

One-hot class vector
(e.g. <0,0,0,1,0>)

Y: output
X: input
W, b: **learned weight**

Each Neuron can be Simple, but connecting them?



Itself can only address linearly separable problem; but connecting them together makes a huge difference

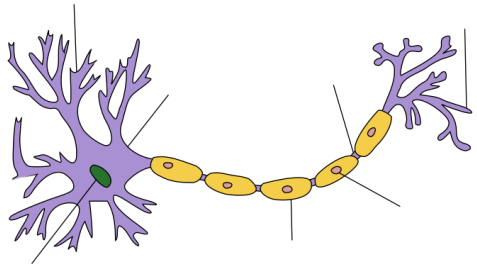
What is Neural Networks?

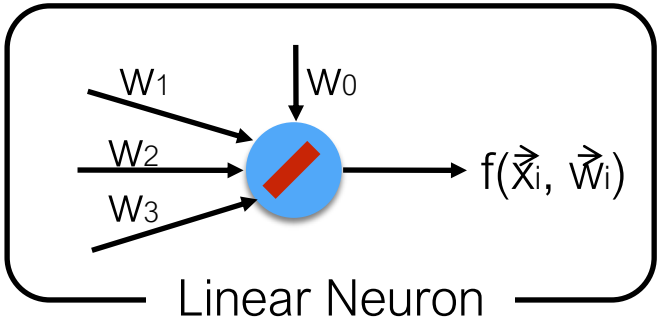
In simple terms,

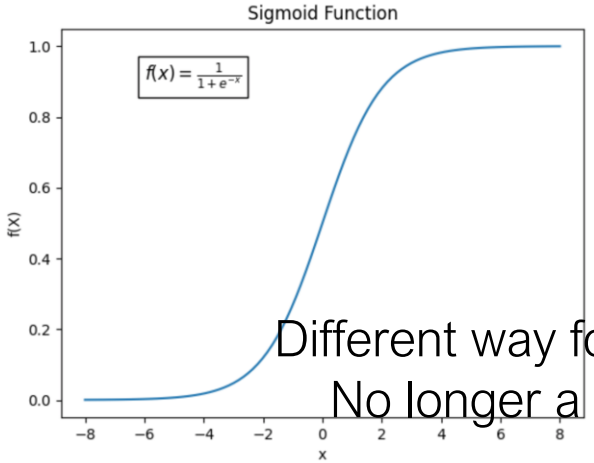
a set of **neurons (atomic functions)**

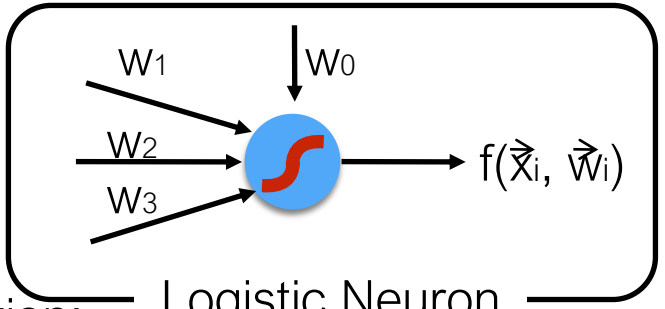
connected in a **non-linear** way (how?)

Neurons



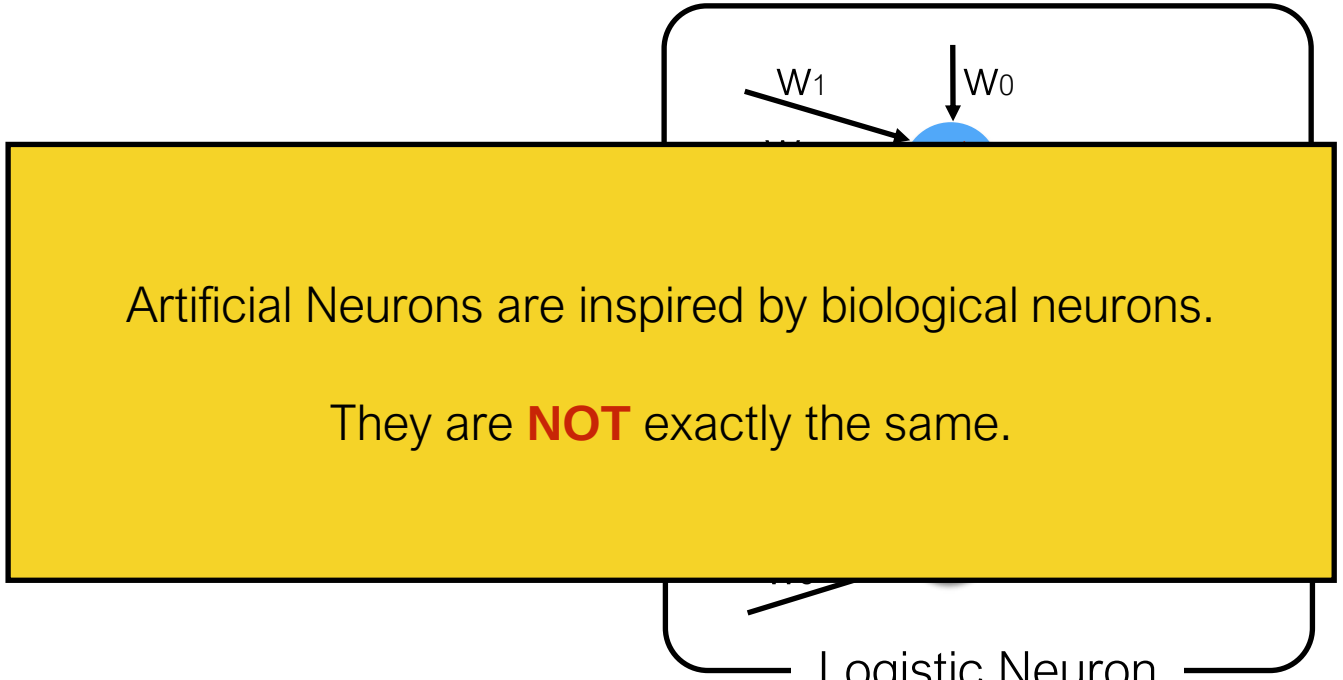






Different way for activation;
No longer a linear one

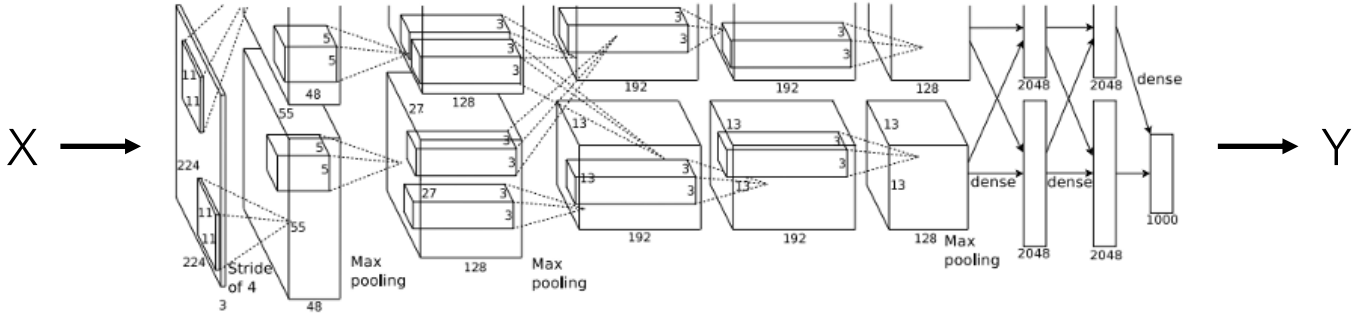
Neurons



More neurons!

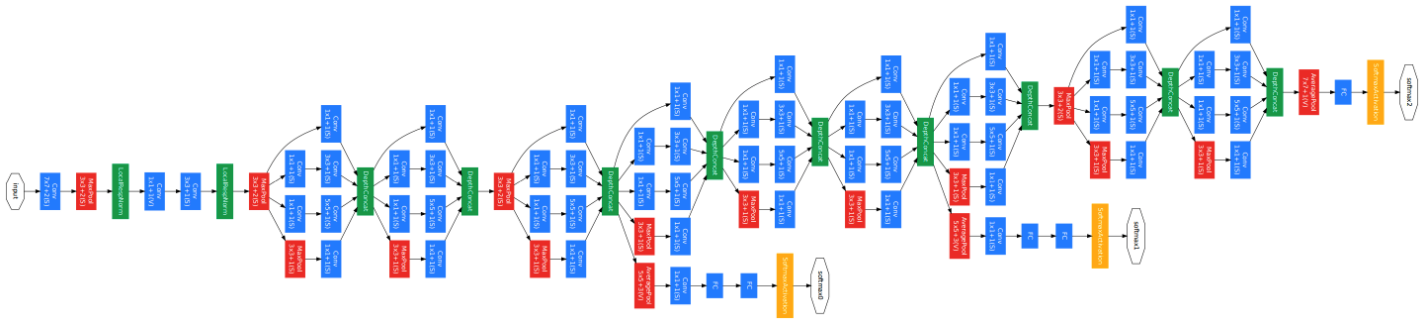
Modified from the HKUST slide

Convolutional Neural Networks



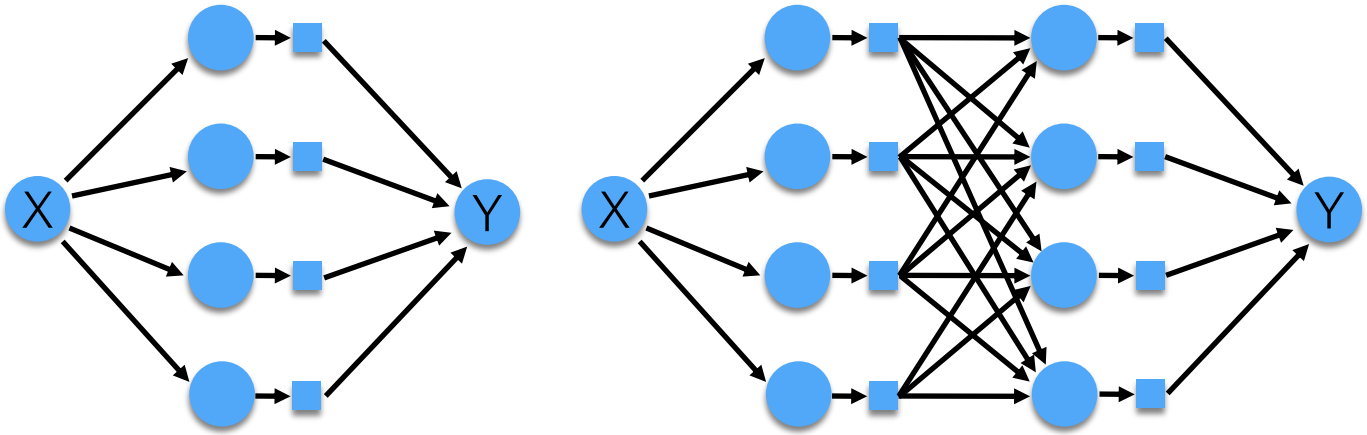
A Krizhevsky, et. al. ImageNet Classification with Deep Convolutional Neural Networks, NIPS 2012

Deep Neural Networks



Szegedy et. al., Going deeper with convolutions, CVPR 2015

Why Deep Neural Networks?

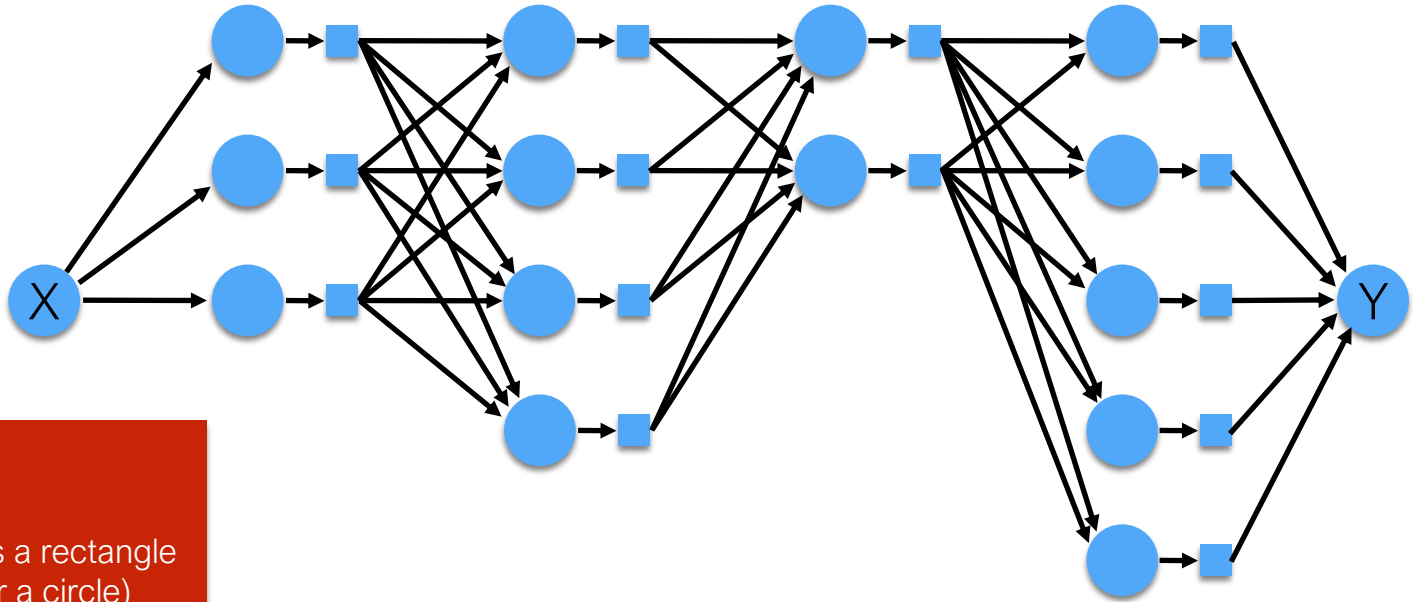


Shallow network

Deep network

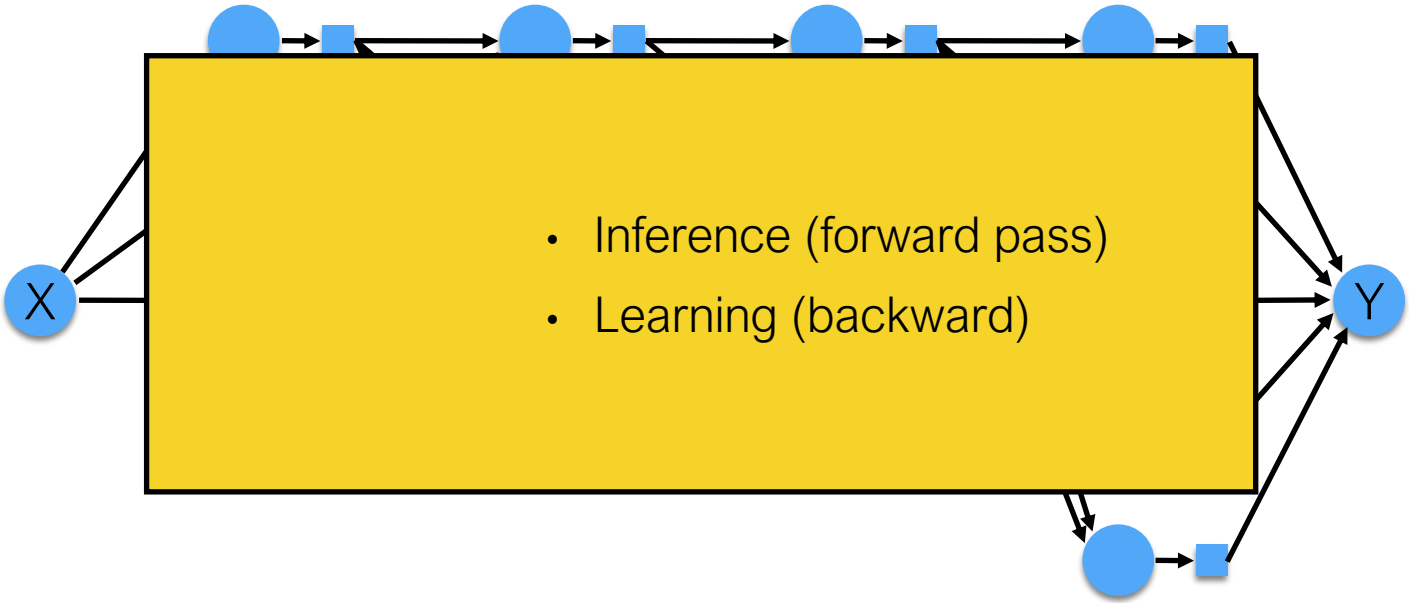
The algorithm for training a **deep** network

Neural Networks Examples

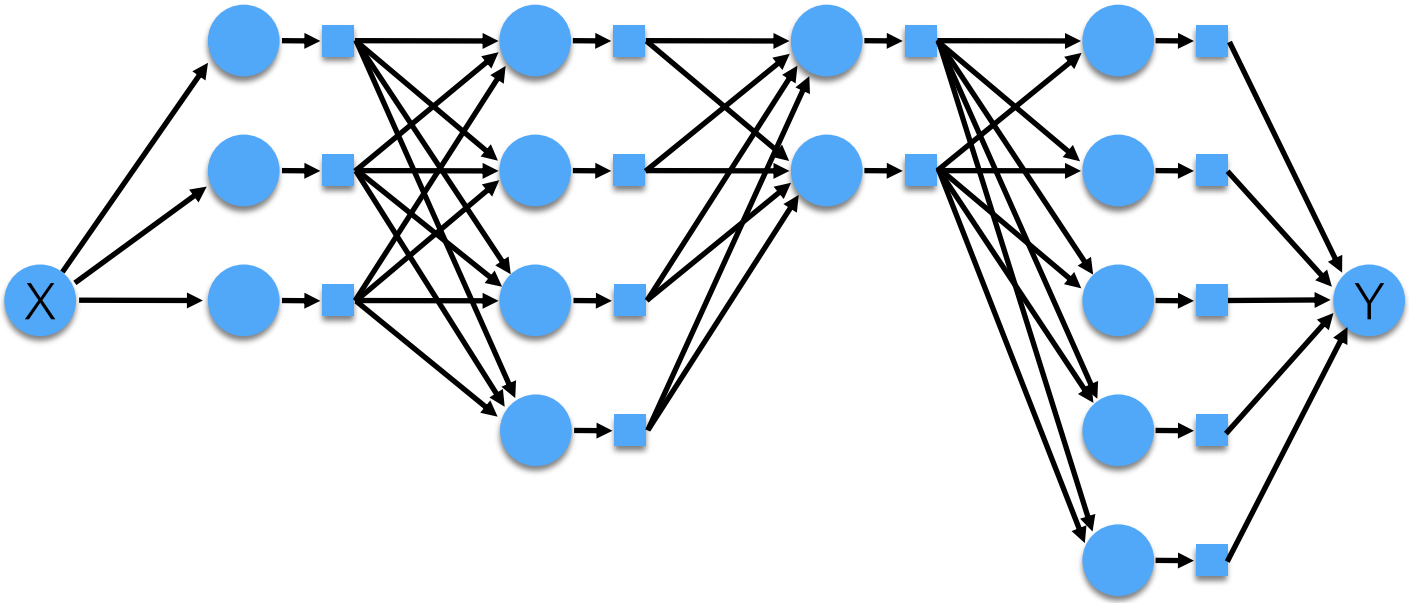


Put Y as a rectangle
(after a circle)

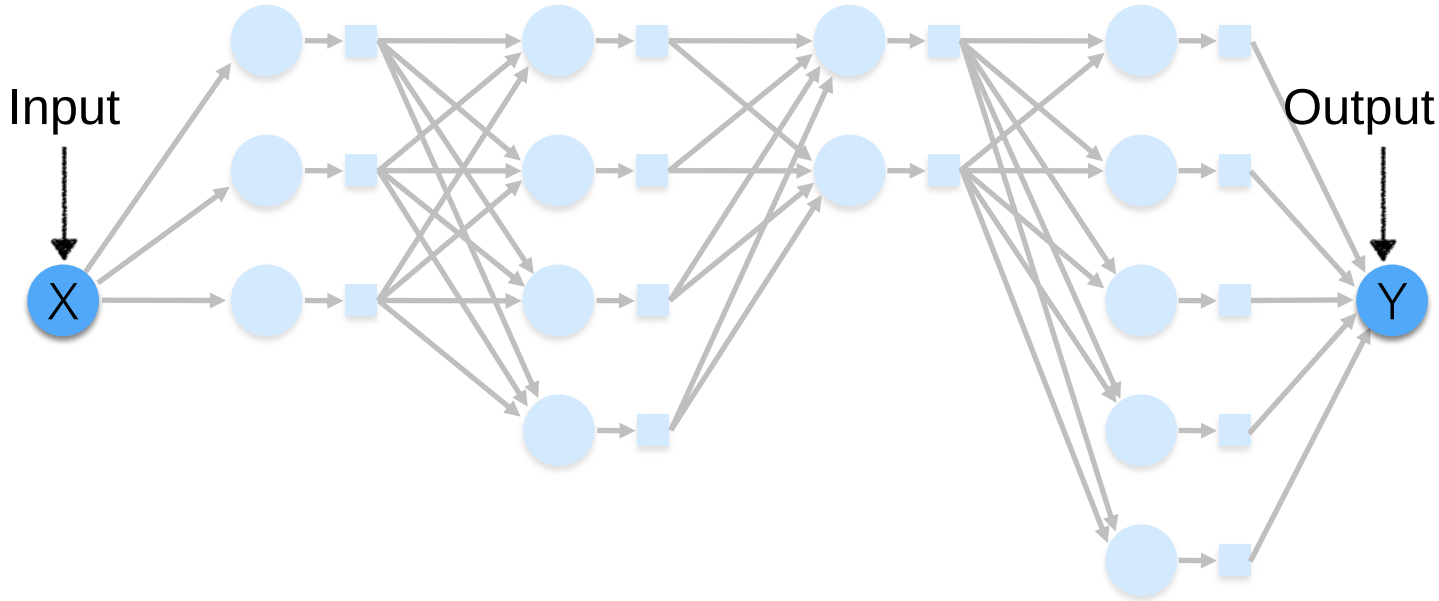
Neural Networks Examples



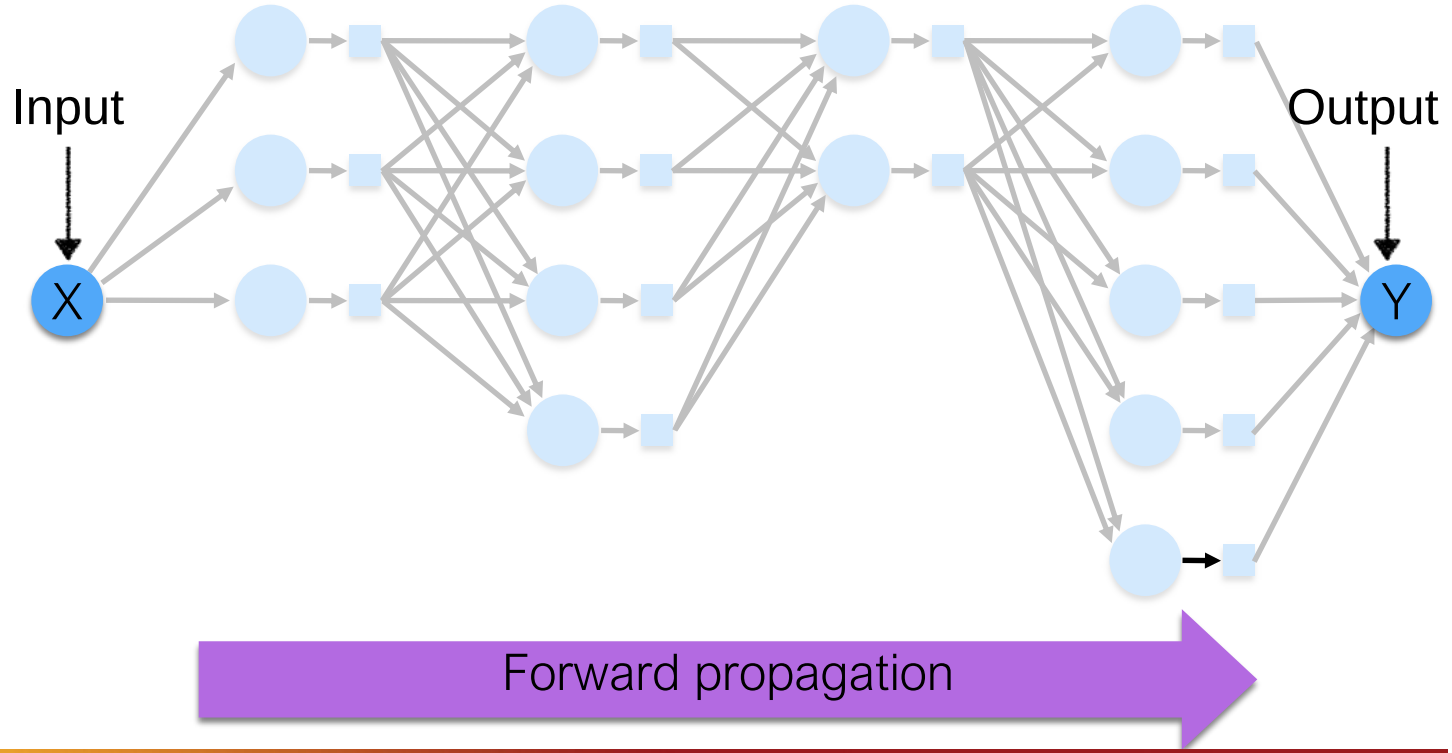
Forward Propagation



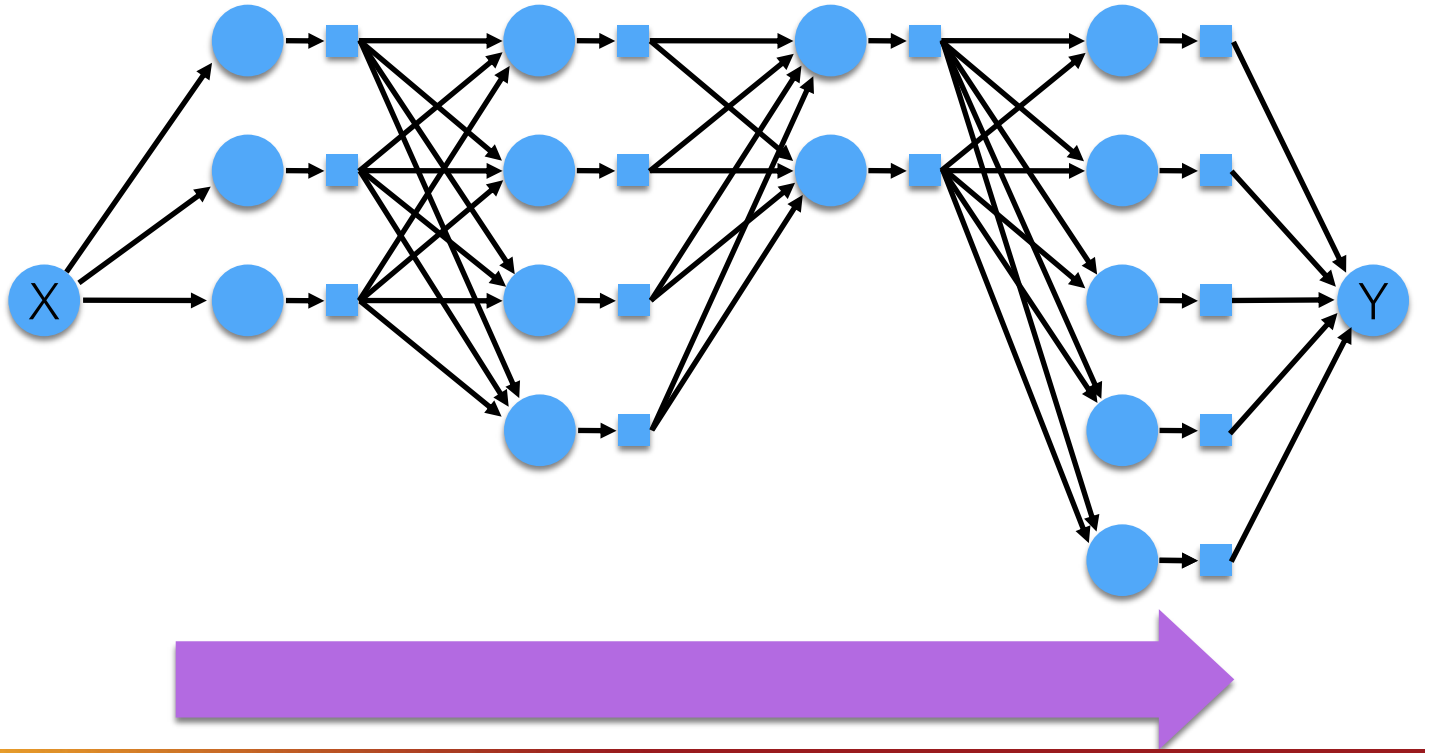
Forward Propagation



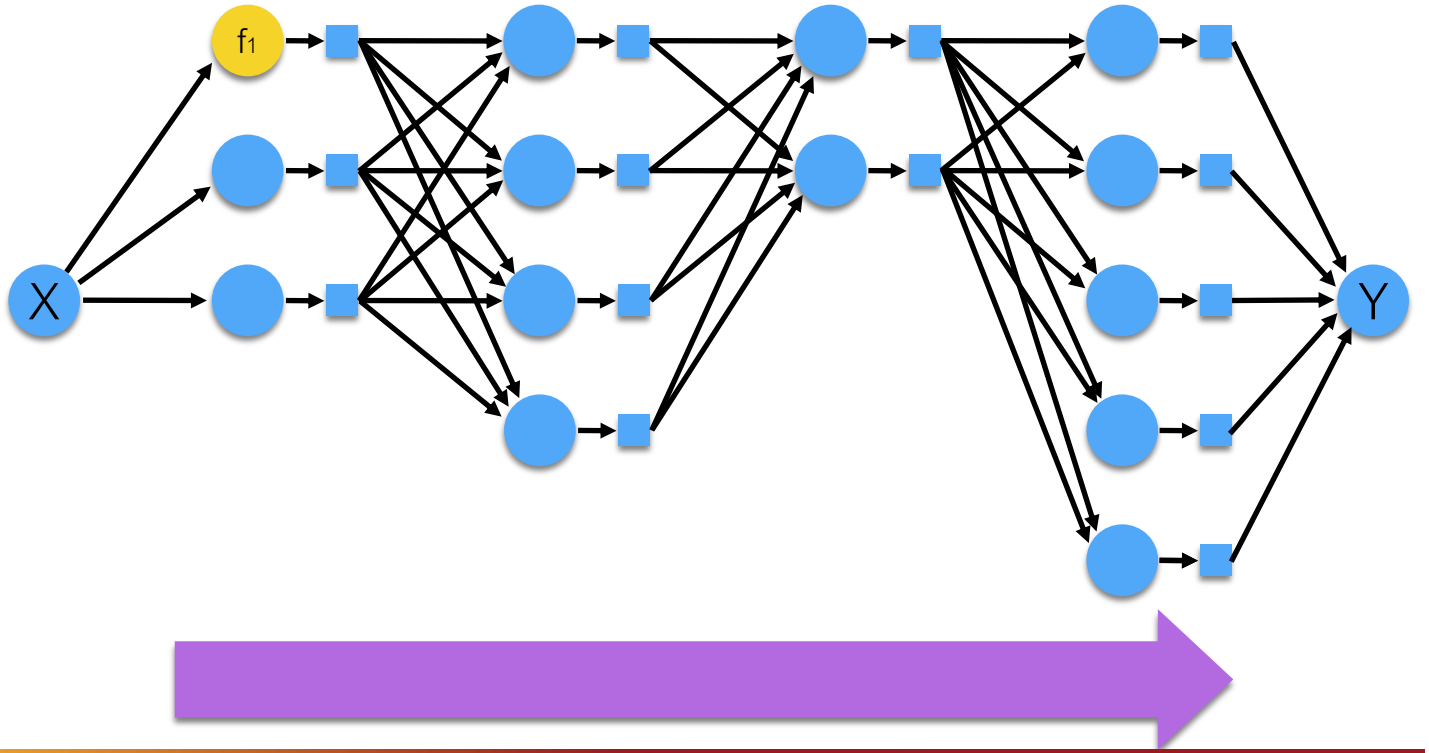
Forward Propagation



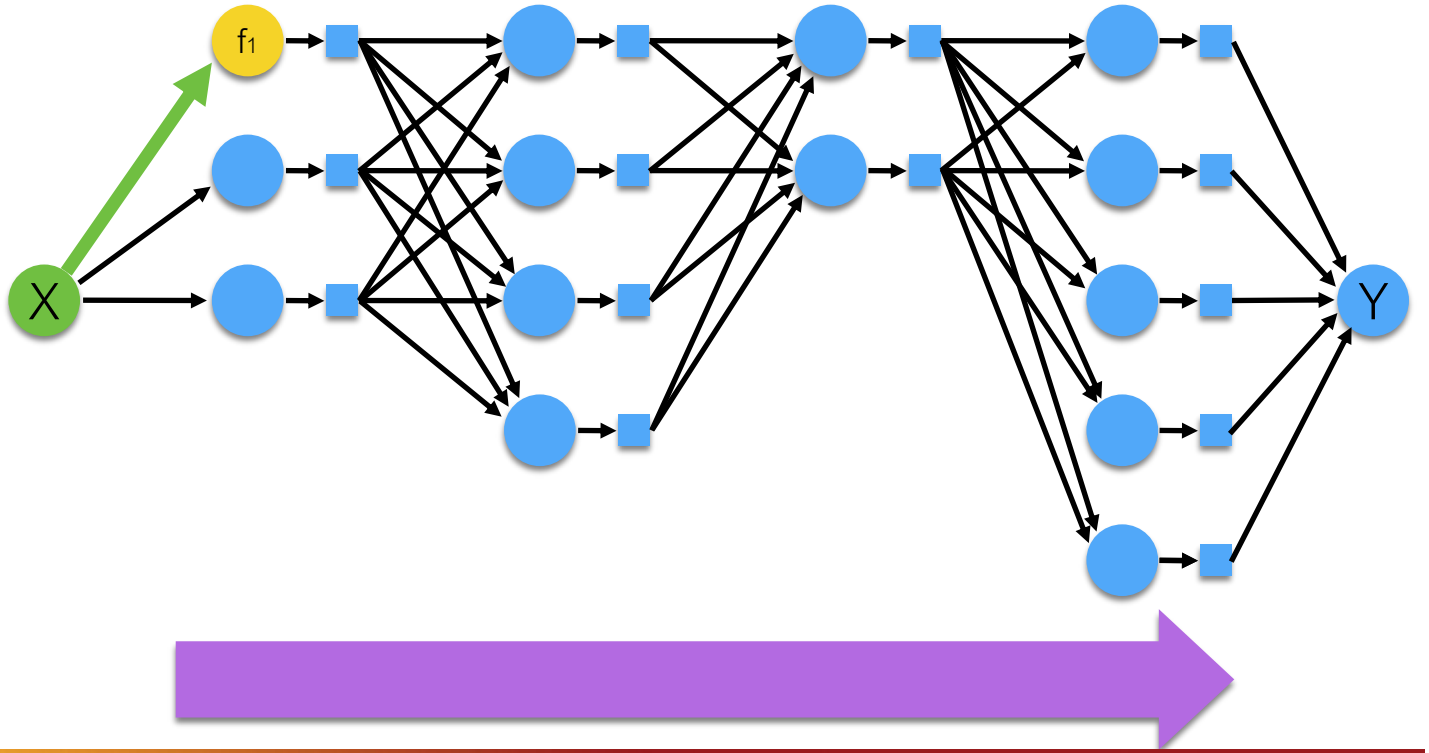
Forward Propagation



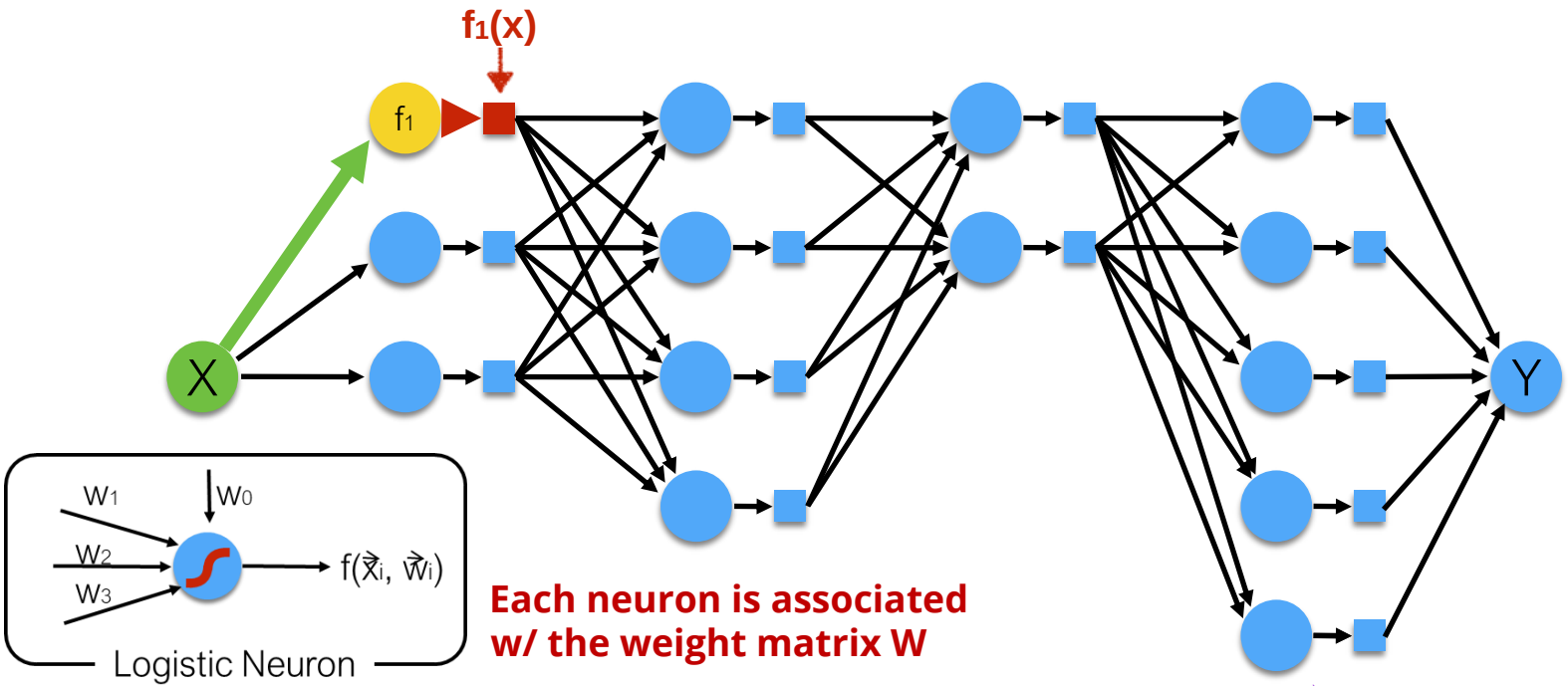
Forward Propagation



Forward Propagation



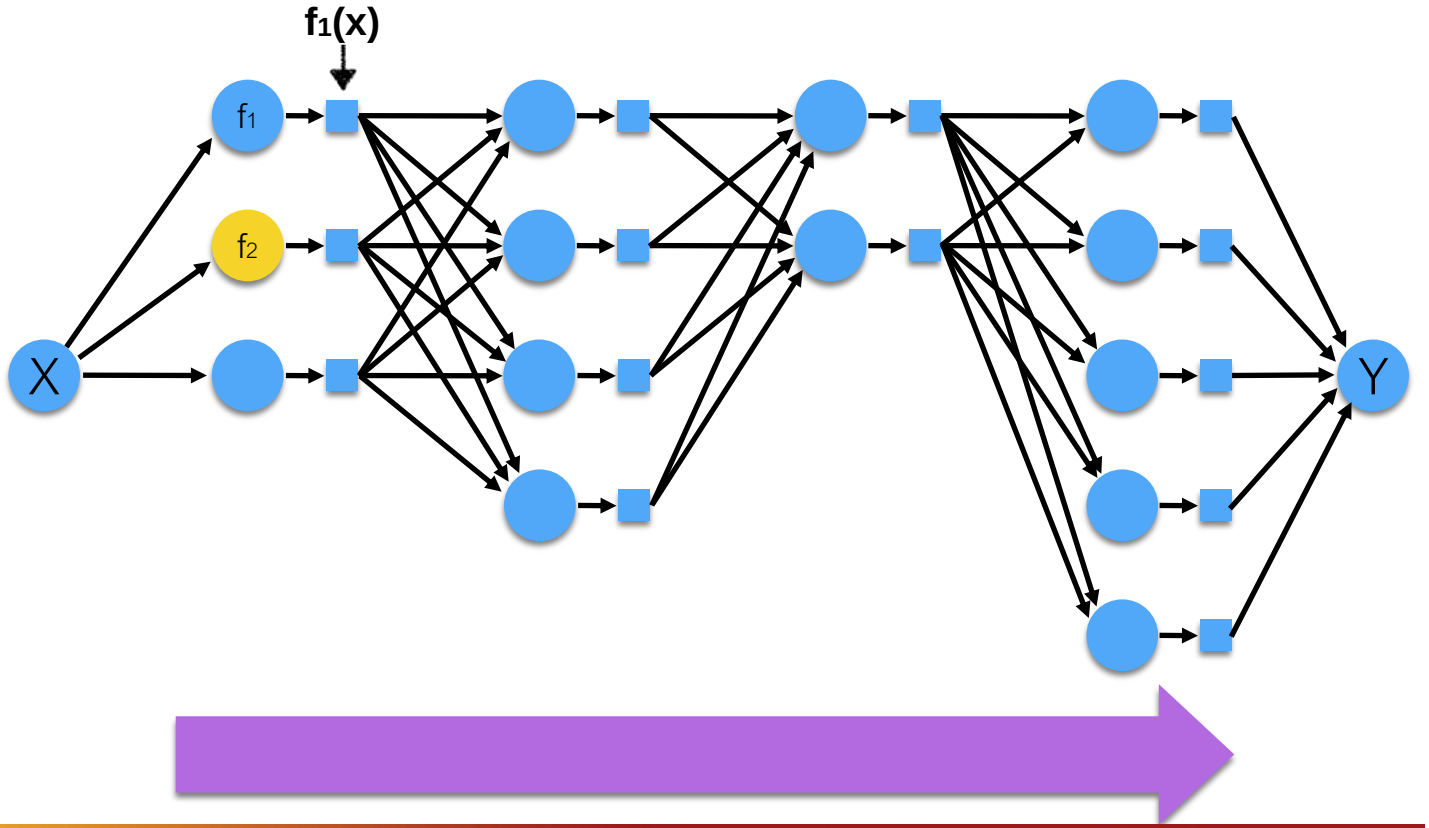
Forward Propagation



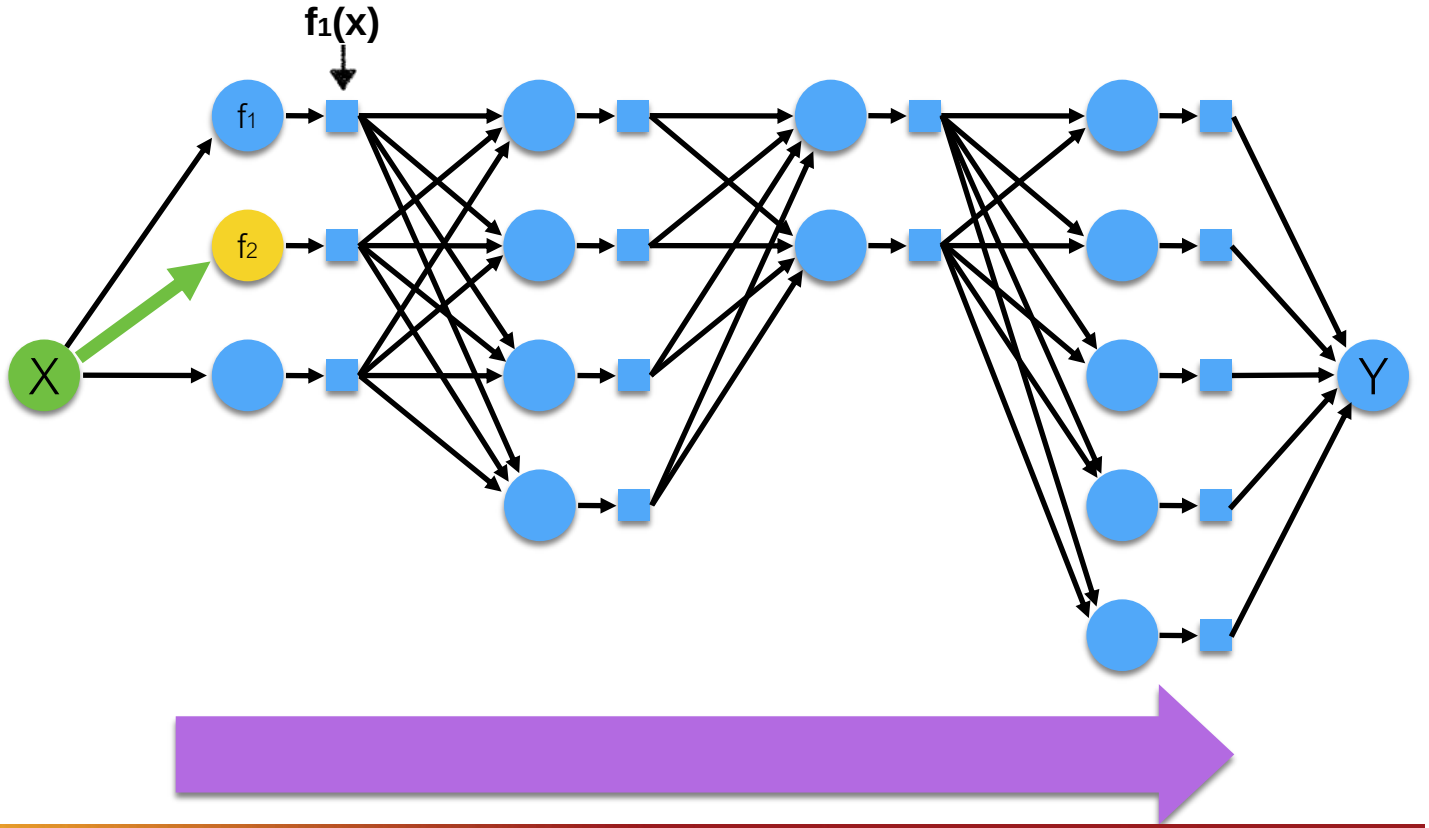
Each neuron is associated w/ the weight matrix W

Sigmod function, can be other non-linear choices

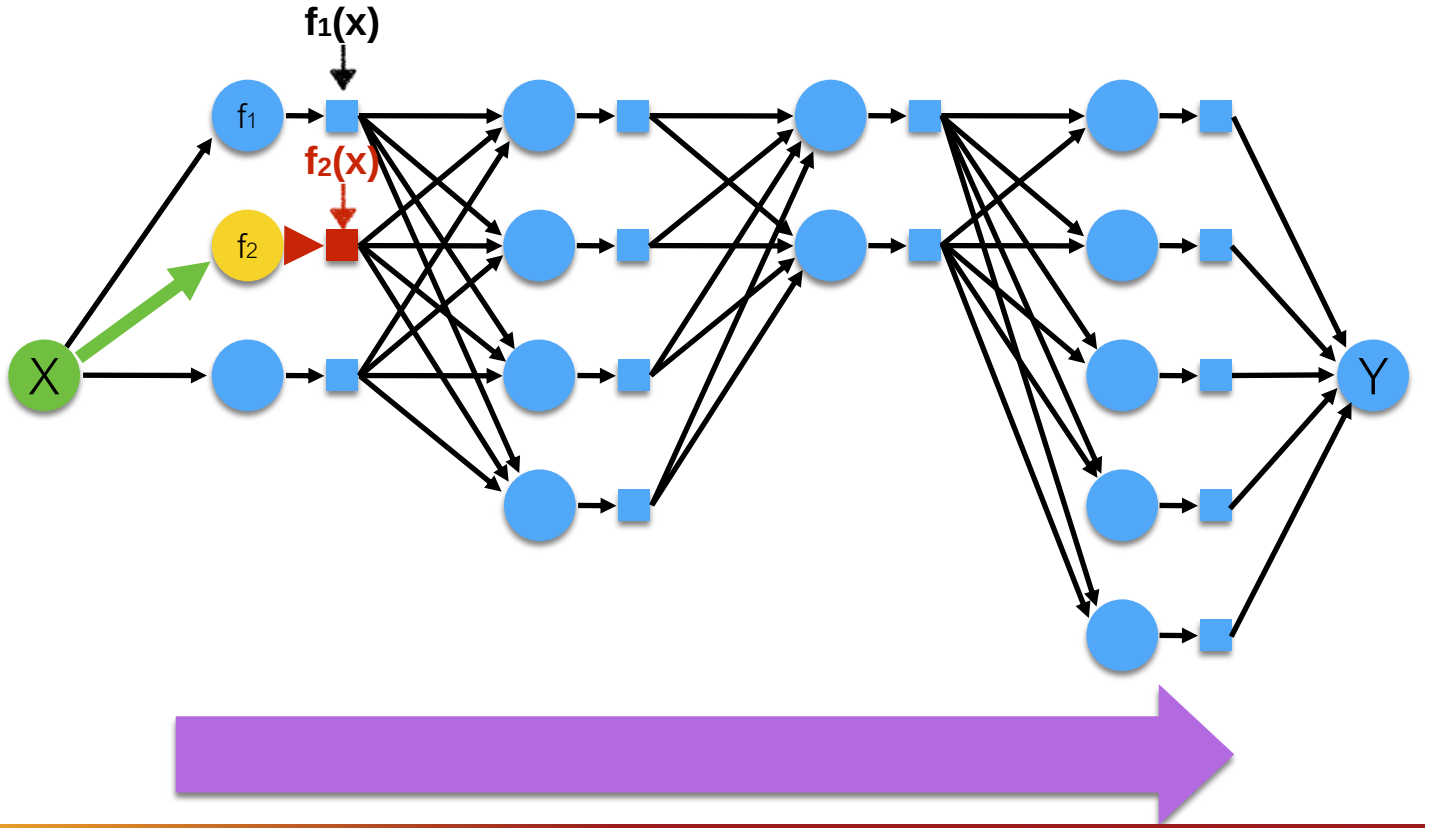
Forward Propagation



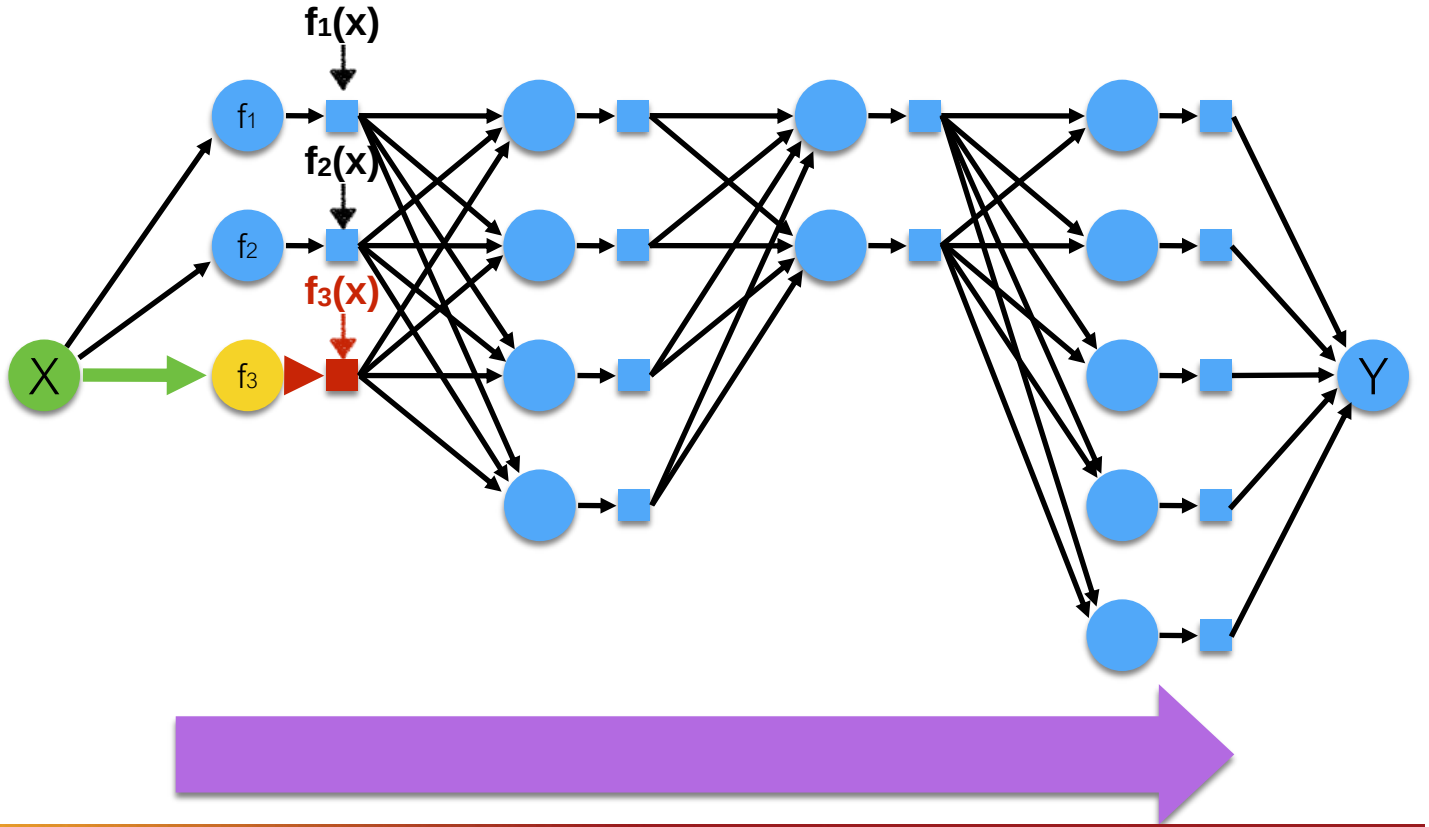
Forward Propagation



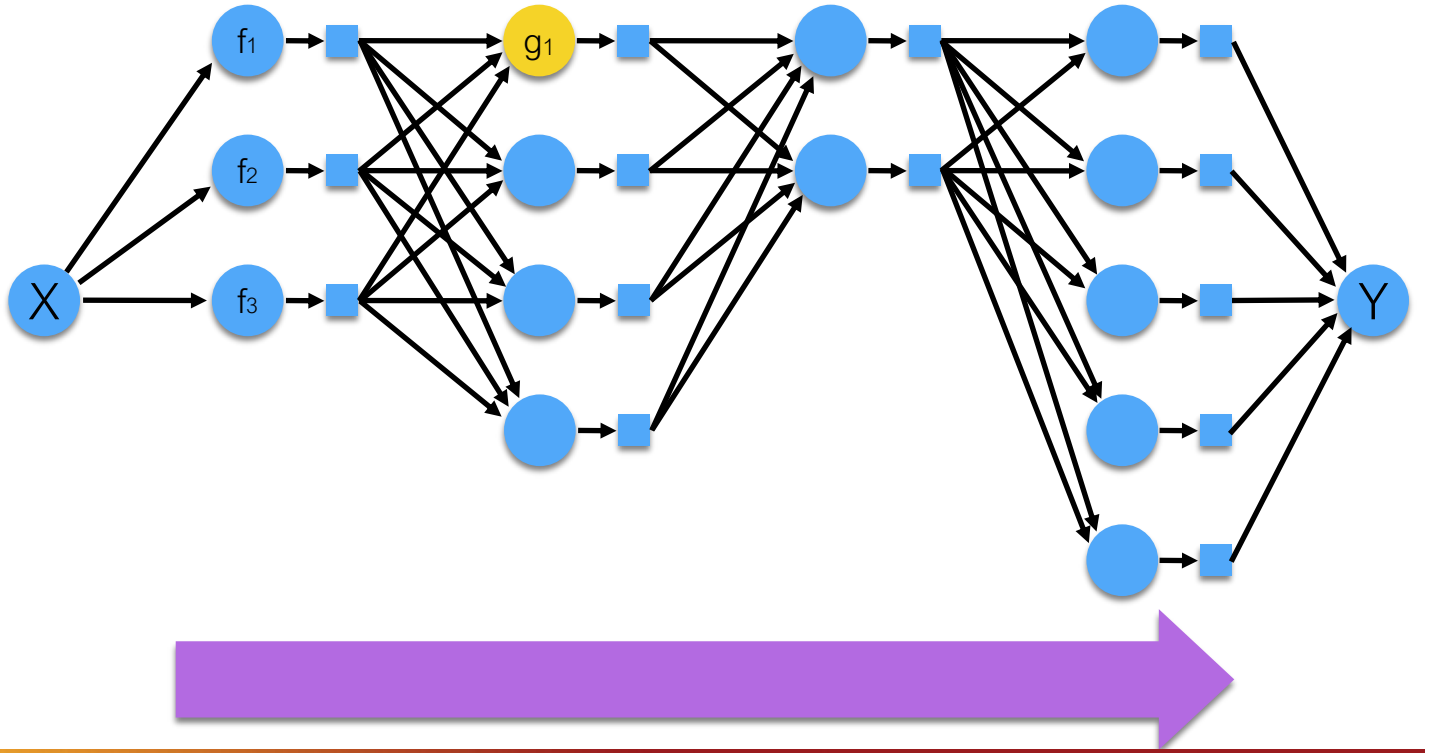
Forward Propagation



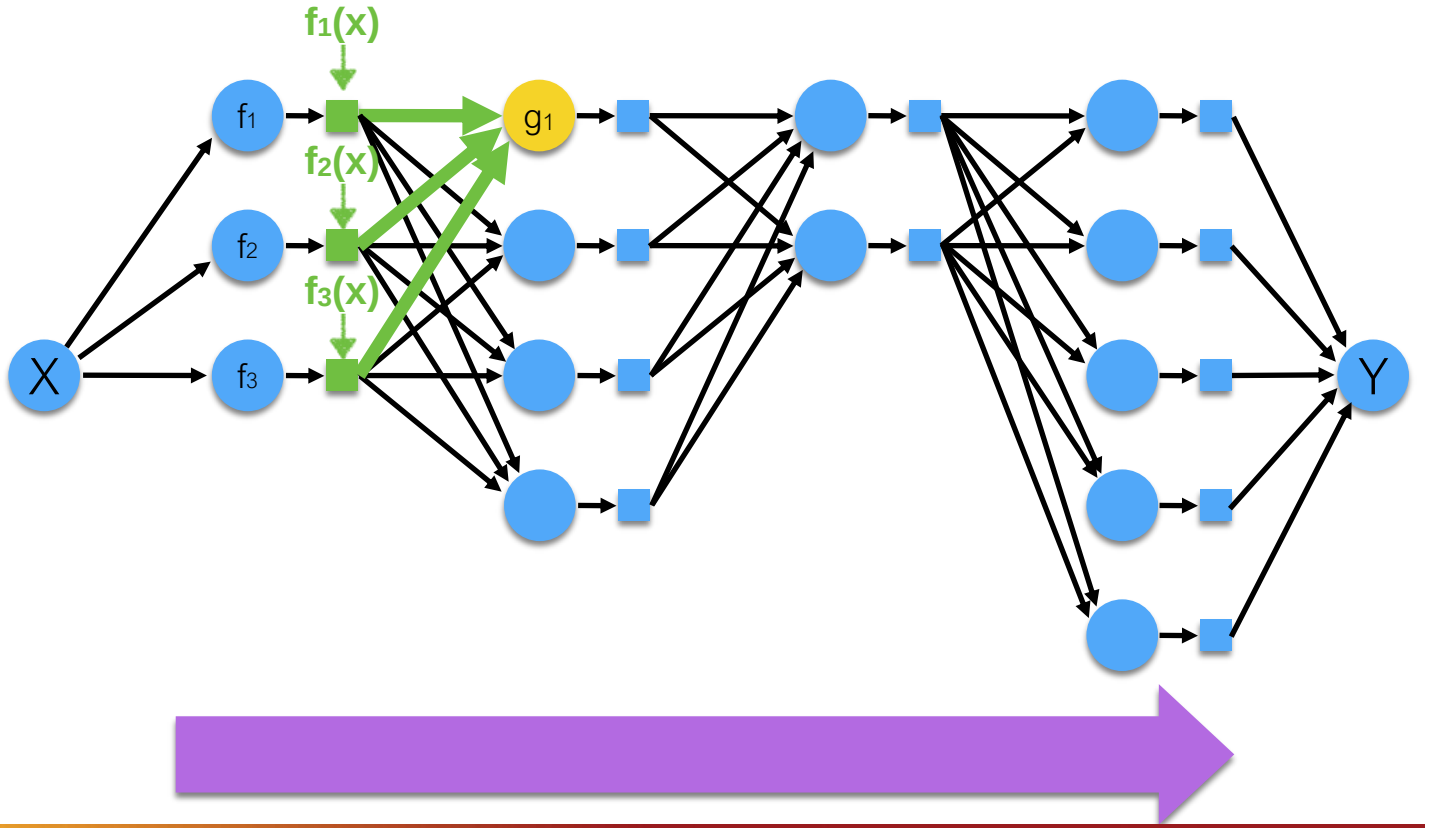
Forward Propagation



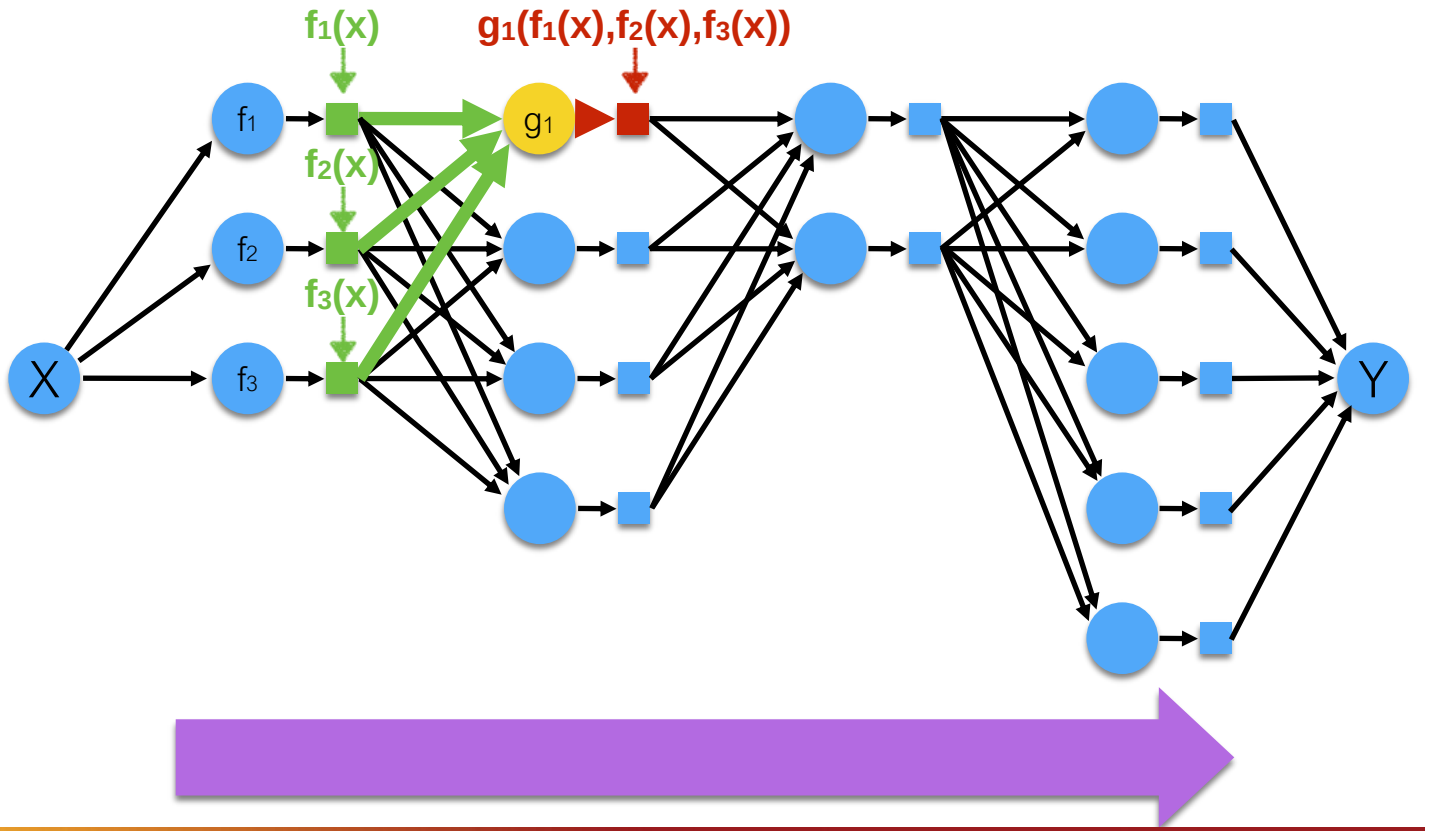
Forward Propagation



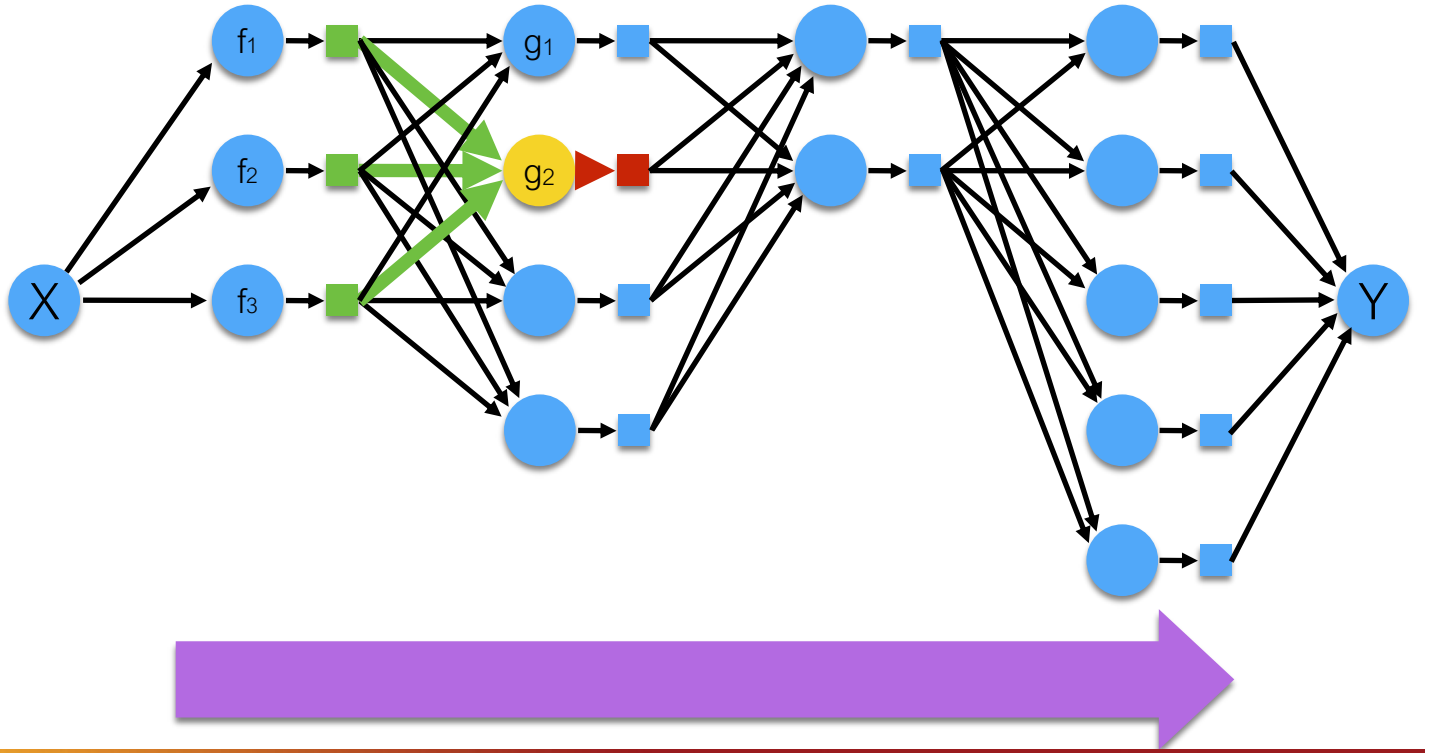
Forward Propagation



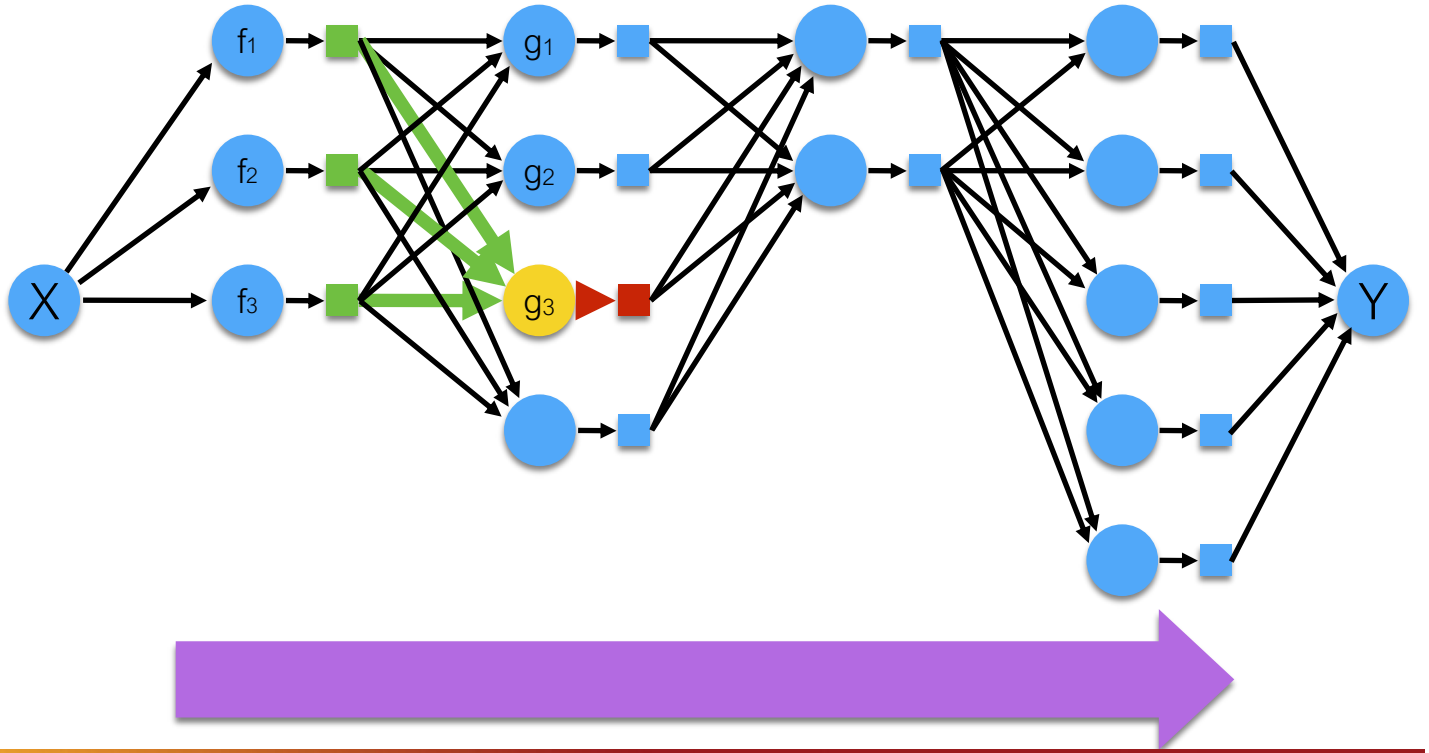
Forward Propagation



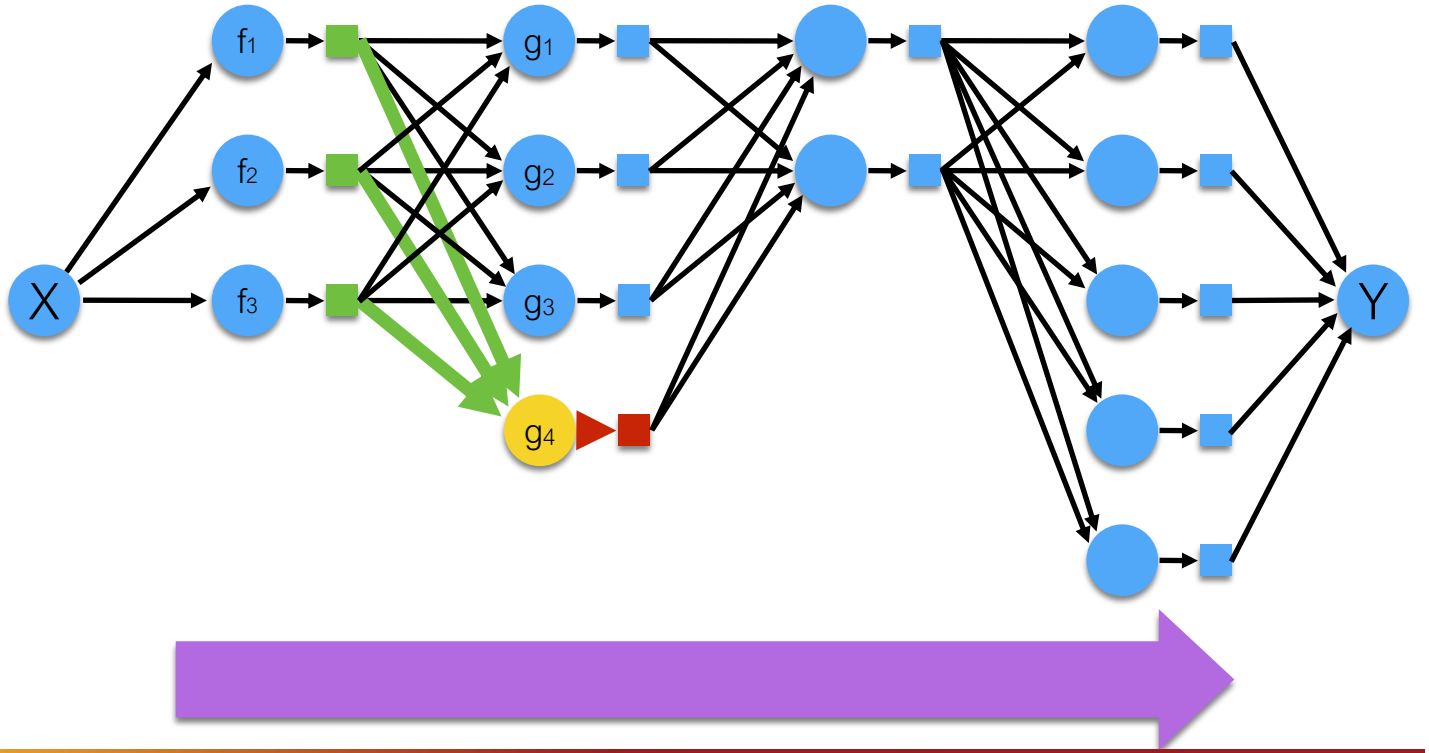
Forward Propagation



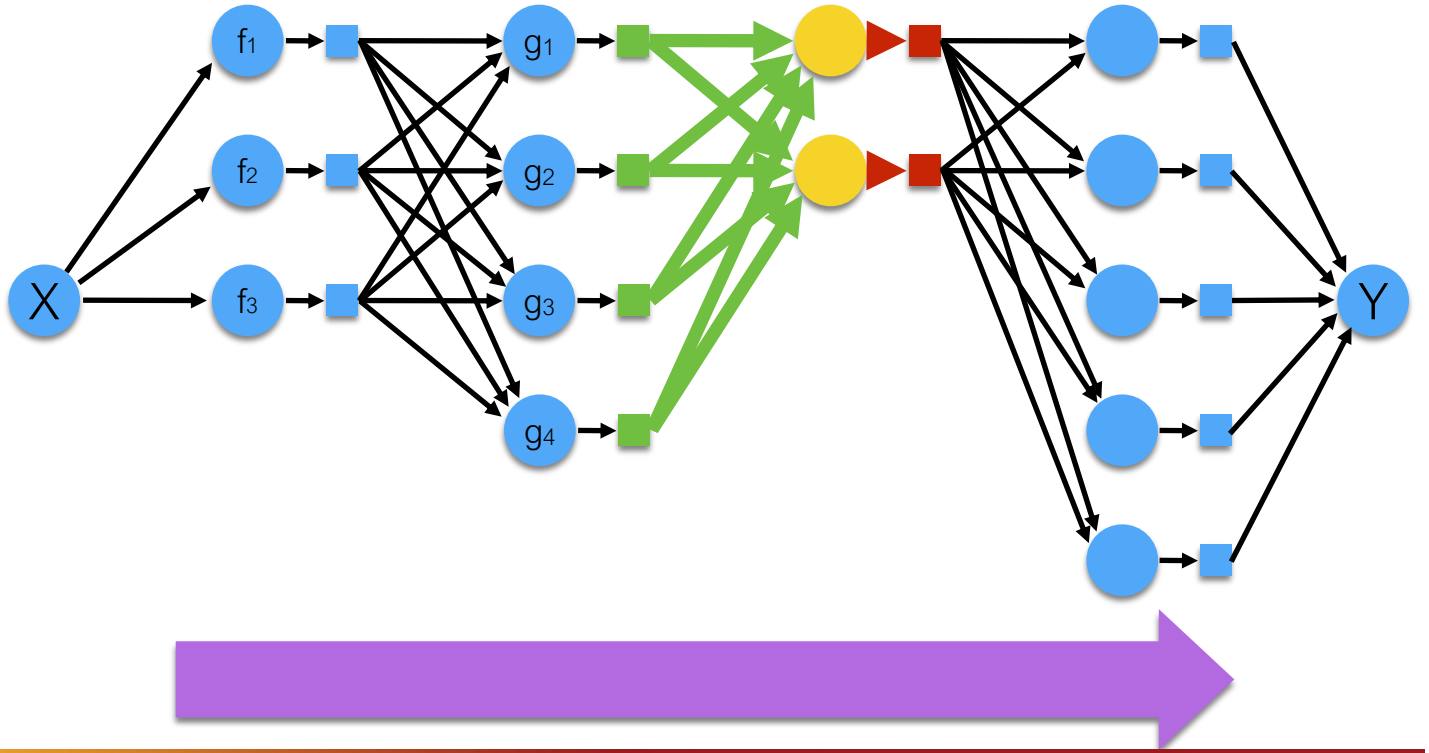
Forward Propagation



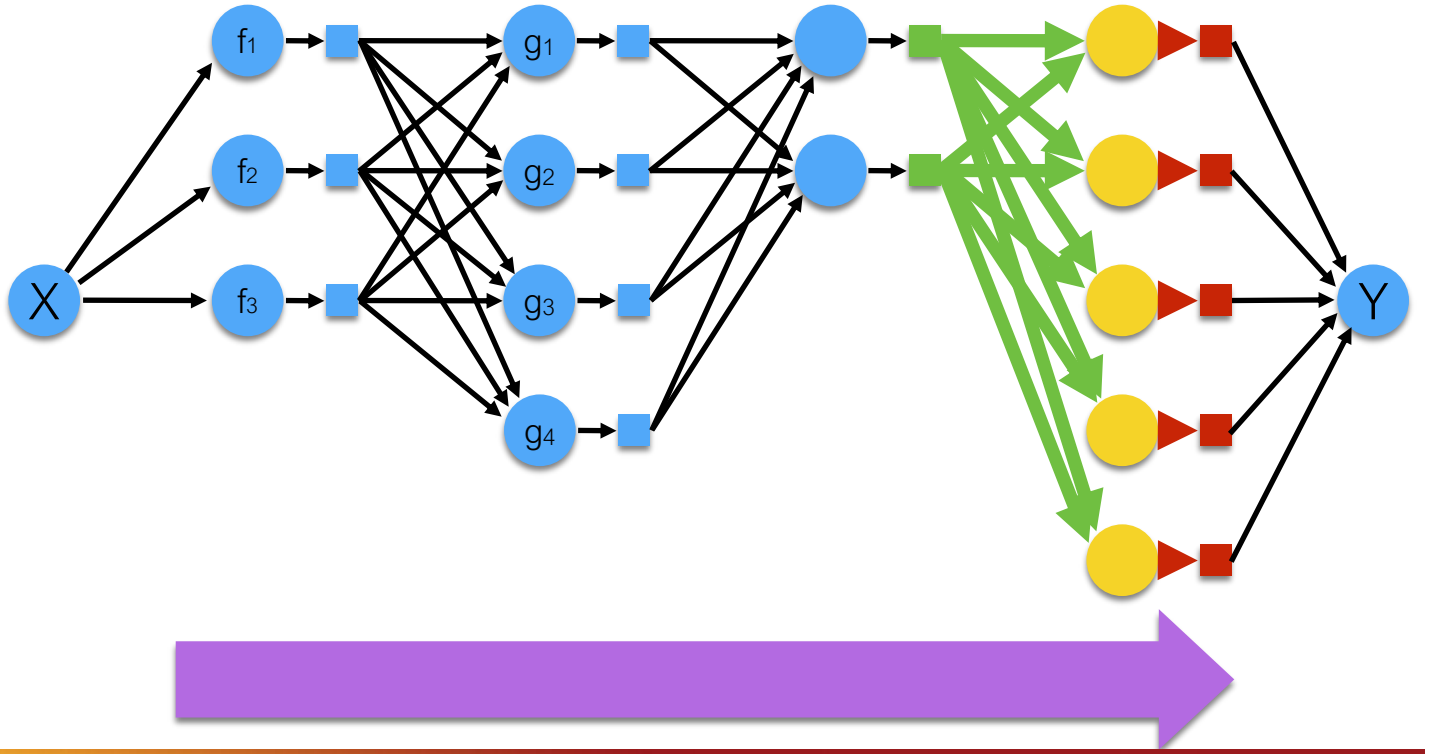
Forward Propagation



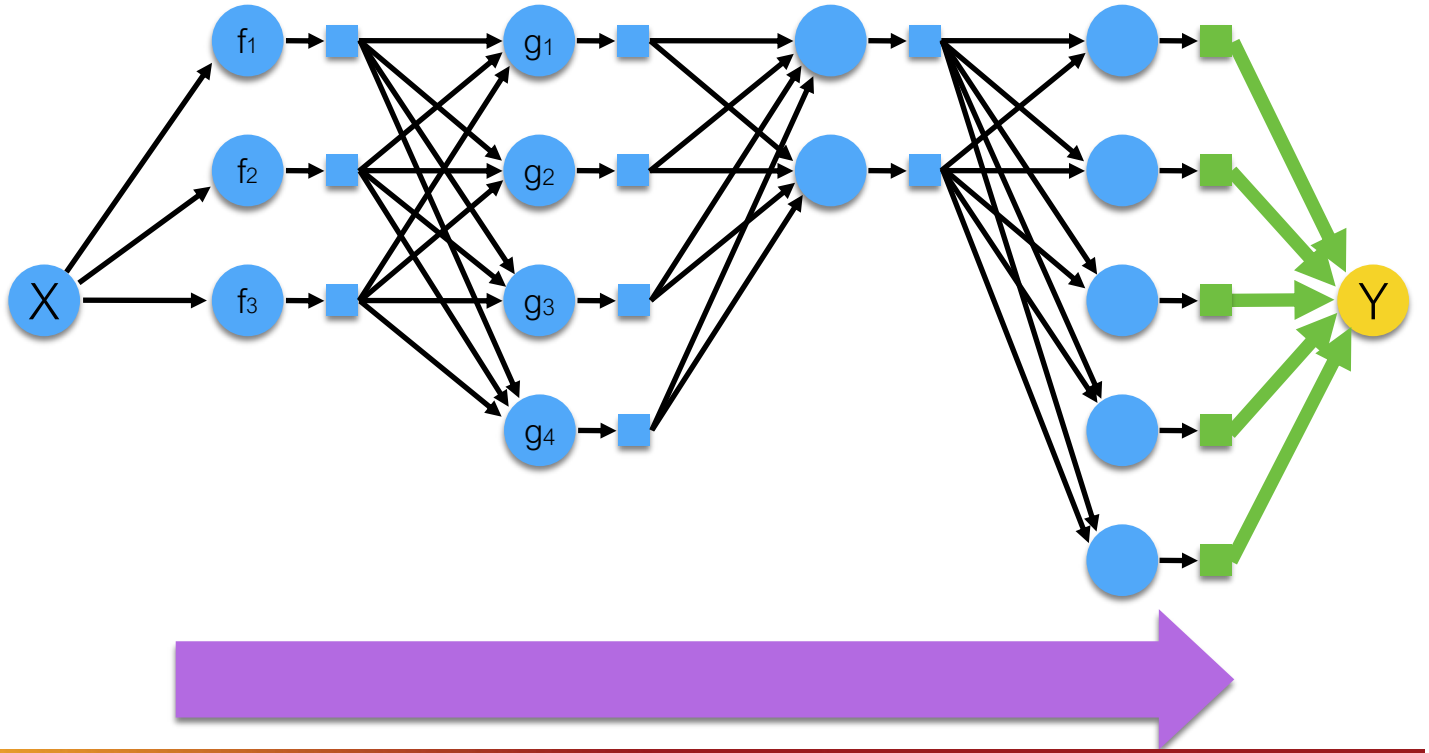
Forward Propagation



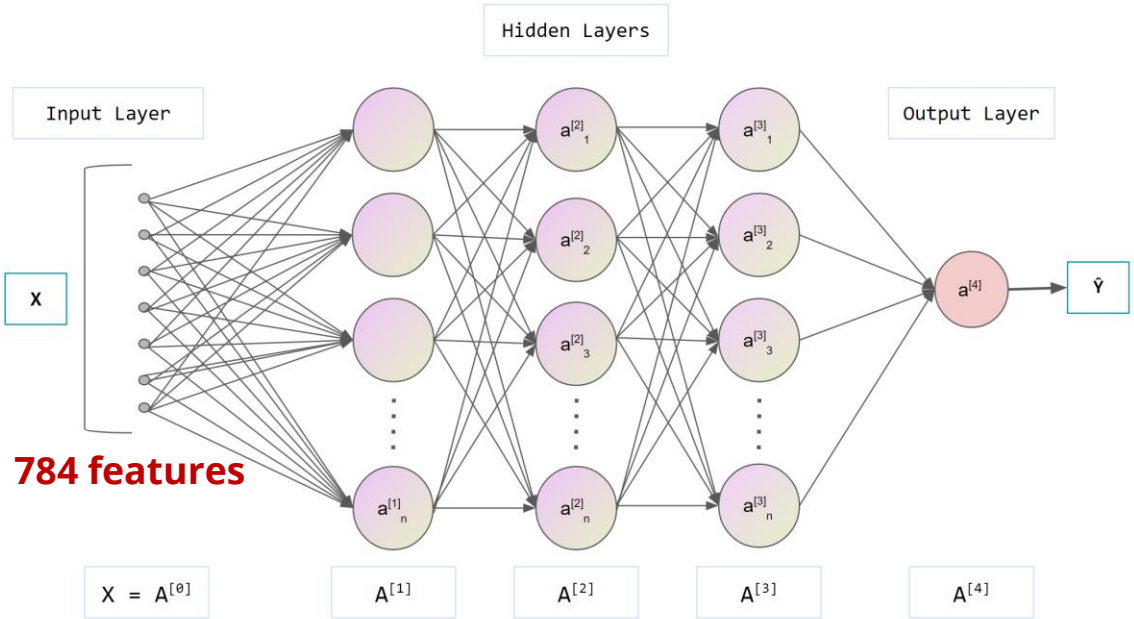
Forward Propagation



Forward Propagation

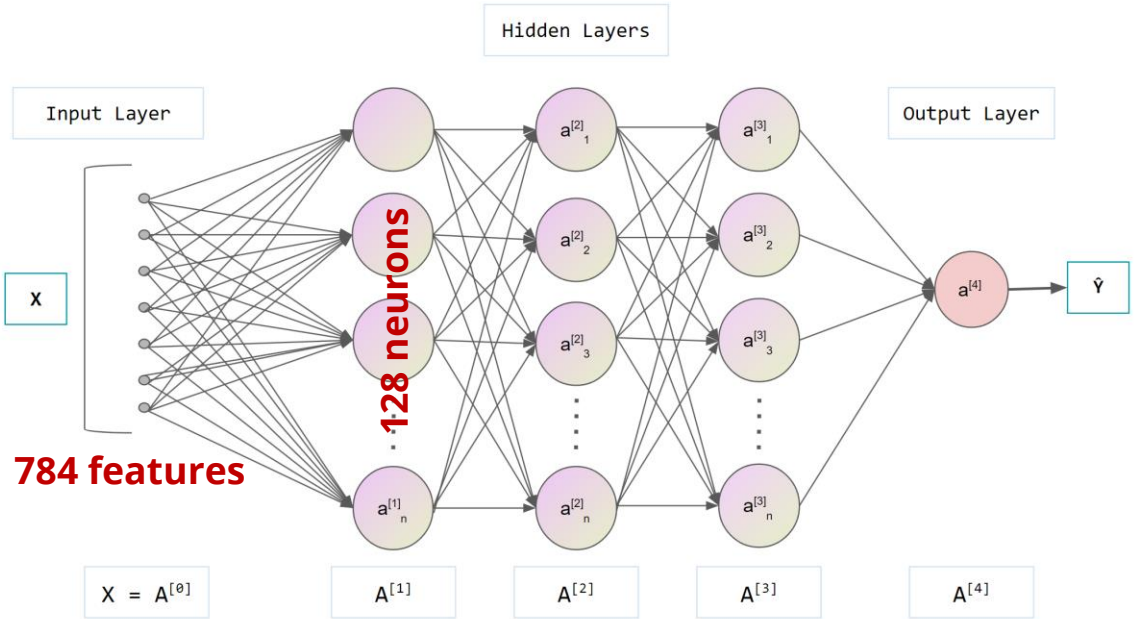


Neural Network Data Shape Explained



Assume each sample is a 28x28 pixels image = 784 features

Neural Network Data Shape Explained



The first hidden layer

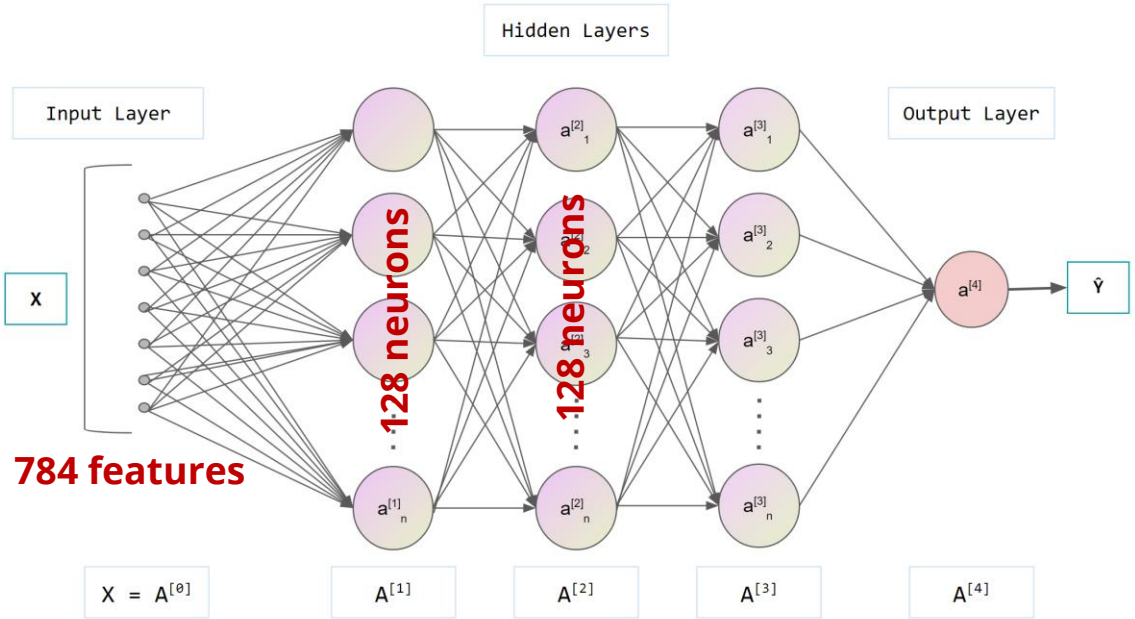
A[1] has 128 neurons

Thus, the weight matrix **W1** for the first layer will be 784x128

After the first layer, it will have

[1x784] · [784x128] = **1x128** outputs

Neural Network Data Shape Explained



After the first layer, it will have

[1x784] · [784x128] = **1x128** outputs

The second layer also gets 128 neurons, and the weight matrix **W2** for the second layer will be 128x128

Neural networks: also called fully connected network

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

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

Activation (can be sigmoid and o

Weight for f2

Weight for f1

$$x \in \mathbb{R}^D, W_1 \in \mathbb{R}^{H \times D}, W_2 \in \mathbb{R}^{C \times H}$$

"Neural Network" is a very broad term; these are more accurately called "fully-connected networks" or sometimes "multi-layer perceptrons" (MLP)

(In practice we will usually add a learnable bias **b** at each layer as well)

Neural networks: 3 layers

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

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

or 3-layer Neural Network $f = W_3 \max(0, W_2 \max(0, W_1 x))$

$$x \in \mathbb{R}^D, W_1 \in \mathbb{R}^{H_1 \times D}, W_2 \in \mathbb{R}^{H_2 \times H_1}, W_3 \in \mathbb{R}^{C \times H_2}$$

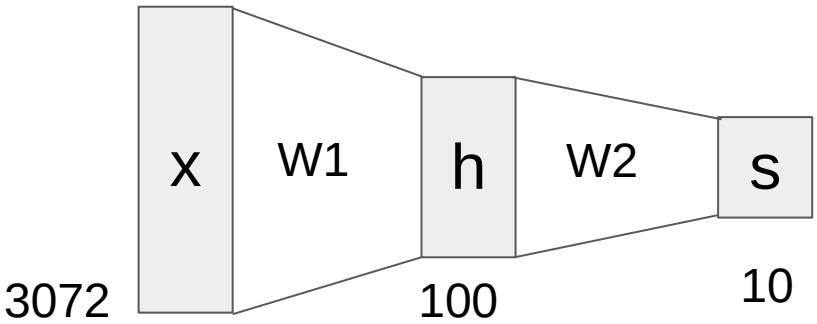
(In practice we will usually add a learnable bias at each layer as well)

Credit to Stanford CS 229

Neural networks: hierarchical computation

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

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



Input features compressed from 3072 to 10

$$x \in \mathbb{R}^D, W_1 \in \mathbb{R}^{H \times D}, W_2 \in \mathbb{R}^{C \times H}$$

Credit to Stanford CS 229

Neural networks: why is max operator important?

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

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

The function $\max(0, z)$ is called the **activation function.**

Q: What if we try to build a neural network without it?

$$f = W_2 W_1 x$$

Credit to Stanford CS 229

Neural networks: why is max operator important?

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

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

The function $\max(0, z)$ is called the **activation function.**

Q: What if we try to build a neural network without one?

$$f = W_2 W_1 x \qquad W_3 = W_2 W_1 \in \mathbb{R}^{C \times H}, f = W_3 x$$

A: We end up with a linear classifier again!

Neural networks need this non linearity!!

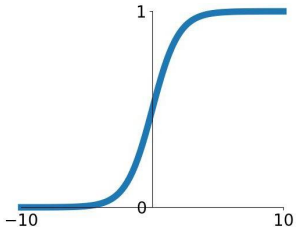
Credit to Stanford CS 229

Activation functions

ReLU is a good default choice for most problems

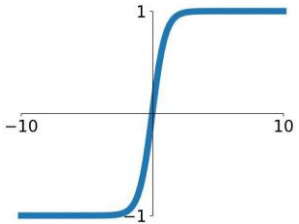
Sigmoid

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



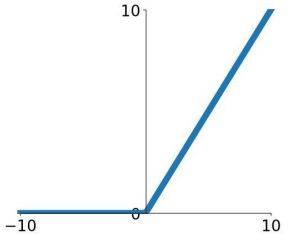
tanh

$$\tanh(x)$$



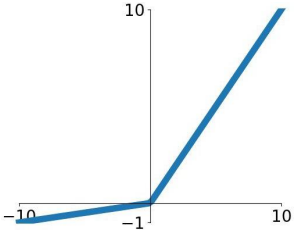
ReLU

$$\max(0, x)$$



Leaky ReLU

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

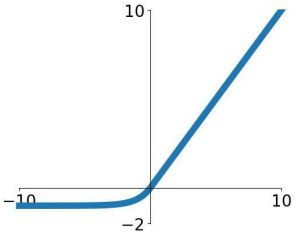


Maxout

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

ELU

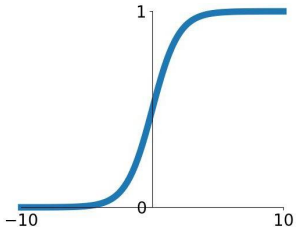
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Activation functions

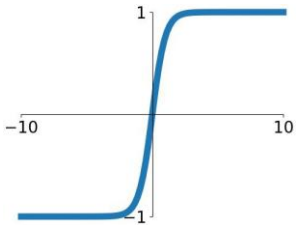
Sigmoid

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

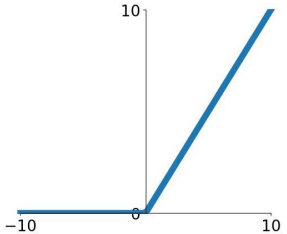


tanh

$$\tanh(x)$$



$$\max(0, x)$$



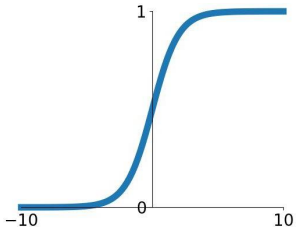
Sigmoid Function

- **Characteristics**: Sigmoid squashes the input values into a range between 0 and 1. It's historically been used for binary classification.
- **When to use**: In the output layer for binary classification tasks. When you need probabilities as outputs, since its output can be interpreted as a probability.
- **Limitations**: Prone to **vanishing gradient problem (discussed later)**. Outputs are not zero-centered.

Activation functions

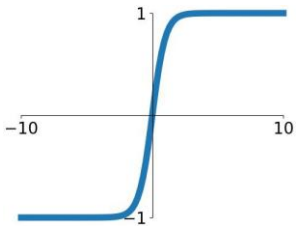
Sigmoid

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



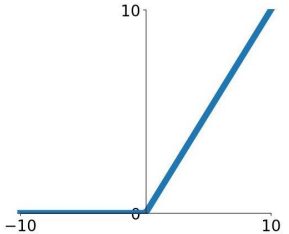
tanh

$$\tanh(x)$$



ReLU

$$\max(0, x)$$



ReLU:

Characteristics: ReLU is a piece-wise linear function that outputs the input directly if positive, otherwise, it will output zero.

When to use: In most hidden layers, as it helps with the vanishing gradient problem and speeds up training. Good default choice for deep neural networks.

Limitations:

Can suffer from "dying ReLU" problem, where neurons can become inactive and only output zero for all inputs.

Activation functions

Leaky ReLU

Characteristics: A variation of ReLU that allows a small, non-zero gradient when the unit is not active.

When to use:

To combat the dying ReLU problem.

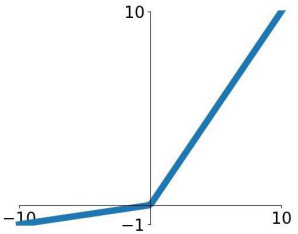
In networks where fast training and convergence are important.

Limitations:

The performance gain over ReLU can be inconsistent.

Leaky ReLU

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

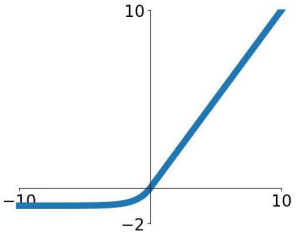


Maxout

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

ELU

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



Important Activation Interview Questions

Question: What is an activation function in neural networks and why are they used?

Important Activation Interview Questions

Question: What is an activation function in neural networks and why are they used?

Answer: An activation function in a neural network is a mathematical function applied to the output of a neuron or layer of neurons, transforming the input signal into an output signal.

It's used for introducing **non-linear** properties to the network. Without non-linearity, the neural network would essentially become a linear regression model, incapable of handling complex data patterns.

Important Activation Interview Questions

Question: Can you compare and contrast sigmoid, tanh, and ReLU activation functions?

Important Activation Interview Questions

Question: Can you compare and contrast sigmoid, tanh, and ReLU activation functions?

Answer: Sigmoid squashes the input values into a range between 0 and 1. It's often used in the output layer for **binary classification**.

Tanh is similar but outputs values between -1 and 1, making it zero-centered and thus, in some cases, more efficient than sigmoid.

ReLU is piece-wise linear, outputting the input directly if it is positive, otherwise zero. It's widely used in hidden layers due to its computational efficiency and ability to mitigate the **vanishing gradient** problem. However, ReLU can suffer from the "dying ReLU" problem, where neurons stop learning completely.

Important Activation Interview Questions

Question: What is the 'dying ReLU' problem and how can it be addressed?

Important Activation Interview Questions

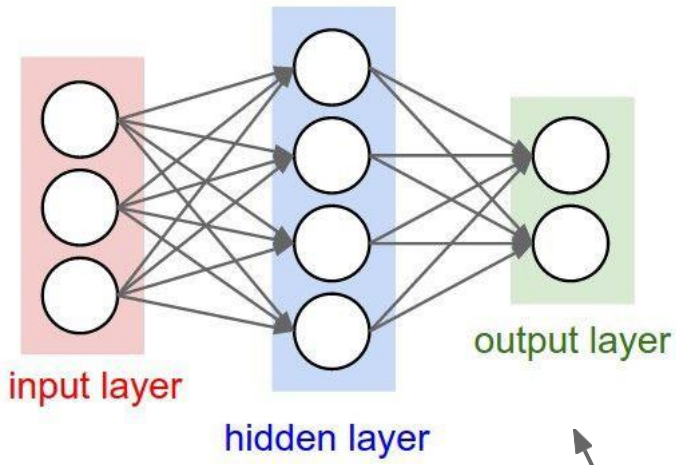
Question: What is the 'dying ReLU' problem and how can it be addressed?

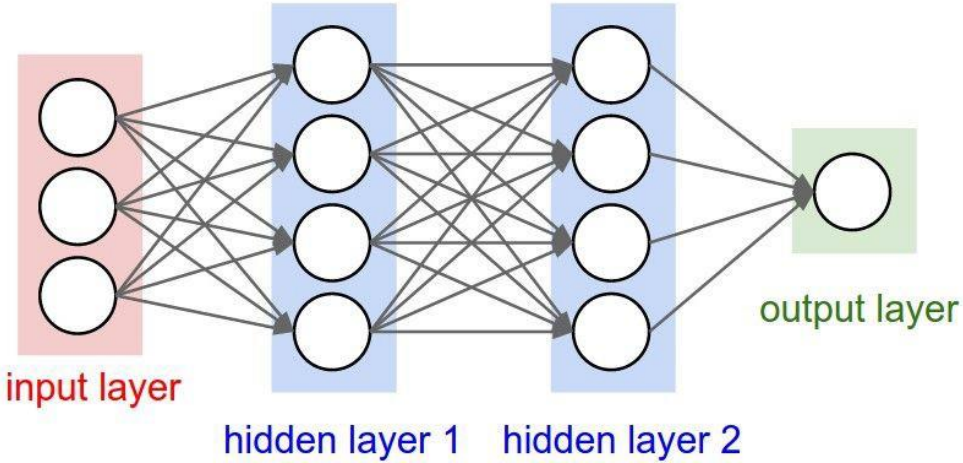
Answer: The 'dying ReLU' problem occurs when **ReLU** neurons become inactive and **only output zero for all inputs**.

This happens when negative inputs shift the neuron's weights in such a way that the neuron never activates on any data point again.

To address this, variants of ReLU like **Leaky ReLU** or Parametric ReLU (PReLU) are used. These functions allow a small, positive gradient when the unit is not active, thereby keeping the neurons alive.

Neural networks: Architectures (Revisited)





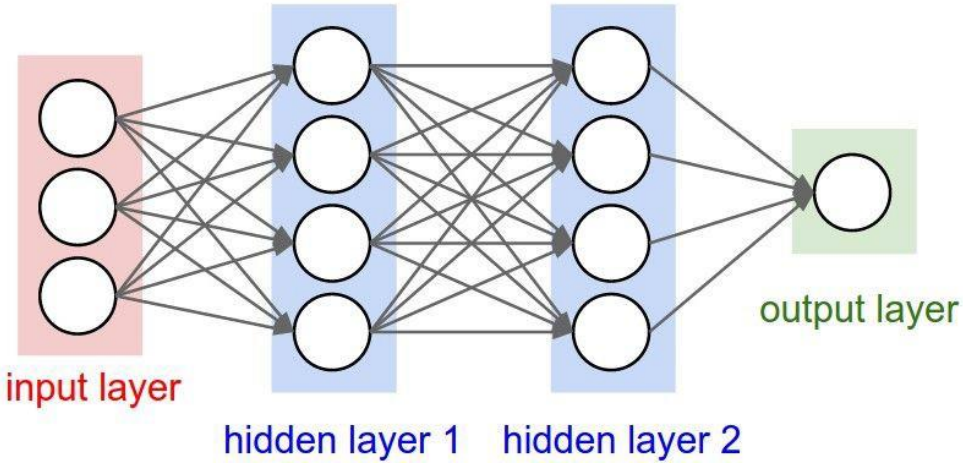
"2-layer Neural Net", or
"1-hidden-layer Neural Net"

"3-layer Neural Net", or
"2-hidden-layer Neural Net"

"Fully-connected" layers

Credit to Stanford CS 229

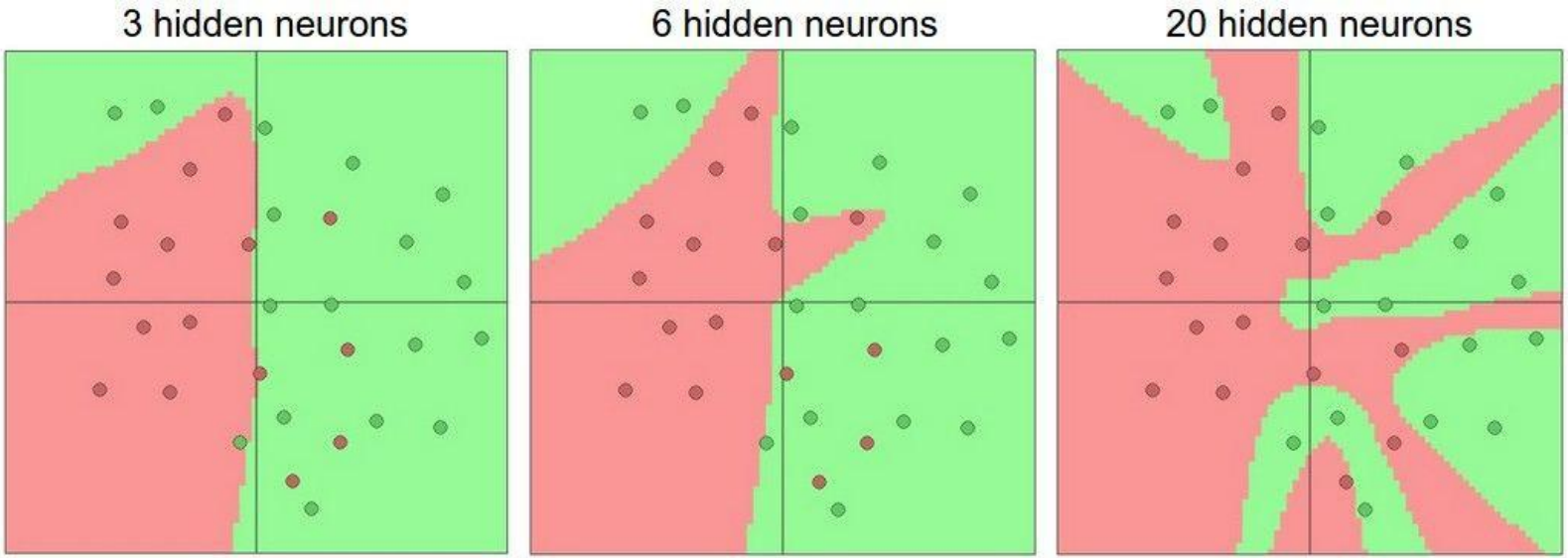
Example feed-forward computation of a neural network




```
# forward-pass of a 3-layer neural network:
f = lambda x: 1.0/(1.0 + np.exp(-x)) # activation function (use sigmoid)
x = np.random.randn(3, 1) # random input vector of three numbers (3x1)
h1 = f(np.dot(W1, x) + b1) # calculate first hidden layer activations (4x1)
h2 = f(np.dot(W2, h1) + b2) # calculate second hidden layer activations (4x1)
out = np.dot(W3, h2) + b3 # output neuron (1x1)
```

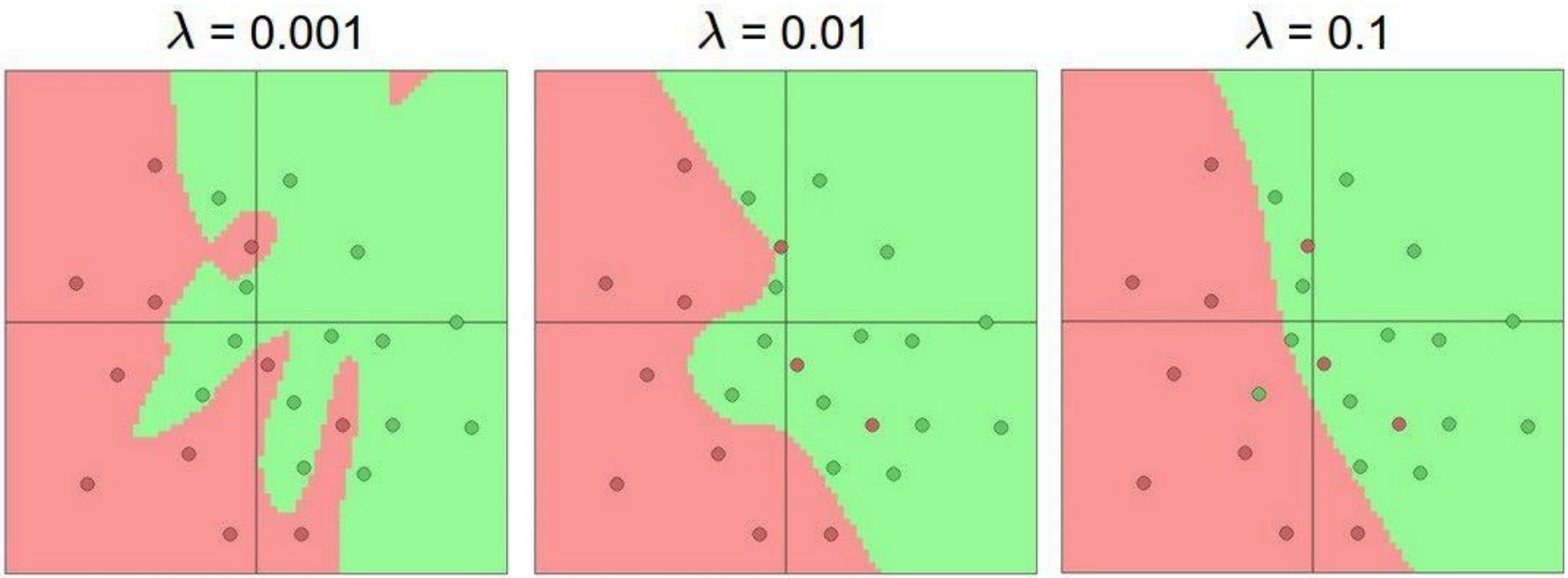
Credit to Stanford CS 229

More Layers Larger Model Capacity



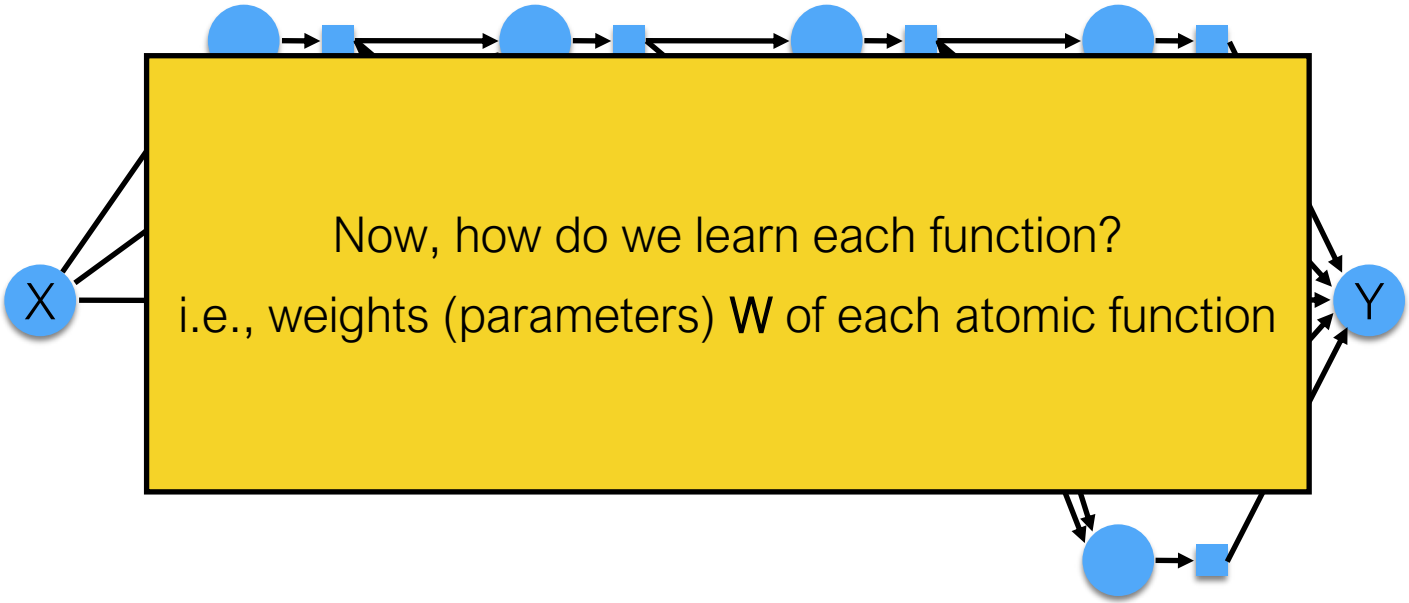
more neurons = more capacity

Larger Model Capacity May Overfit – Need to Regularize them



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

Neural Network Examples



Plugging in neural networks with loss functions

$$s = f(x; W_1, W_2) = W_2 \max(0, W_1 x)$$ Nonlinear score function

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$ Loss on predictions

$$R(W) = \sum_k W_k^2$$ Regularization

$$L = \frac{1}{N} \sum_{i=1}^{N} L_i + \lambda R(W_1) + \lambda R(W_2)$$ Total loss: data loss + regularization

Credit to Stanford CS 231n

Problem: How to compute gradients?

$$s = f(x; W_1, W_2) = W_2 \max(0, W_1 x)$$ Nonlinear score function

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$ SVM Loss on predictions

$$R(W) = \sum_k W_k^2$$ Regularization

$$L = \frac{1}{N} \sum_{i=1}^{N} L_i + \lambda R(W_1) + \lambda R(W_2)$$ Total loss: data loss + regularization

If we can compute $\frac{\partial L}{\partial W_1}, \frac{\partial L}{\partial W_2}$ then we can learn W_1 and W_2

Credit to Stanford CS 231n

(Bad) Idea: Derive $\nabla_W L$ on paper

$$s = f(x; W) = Wx$$

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

$$= \sum_{j \neq y_i} \max(0, W_{j,:} \cdot x + W_{y_i,:} \cdot x + 1)$$

$$L = \frac{1}{N} \sum_{i=1}^{N} L_i + \lambda \sum_k W_k^2$$

$$= \frac{1}{N} \sum_{i=1}^{N} \sum_{j \neq y_i} \max(0, W_{j,:} \cdot x + W_{y_i,:} \cdot x + 1) + \lambda \sum_k W_k^2$$

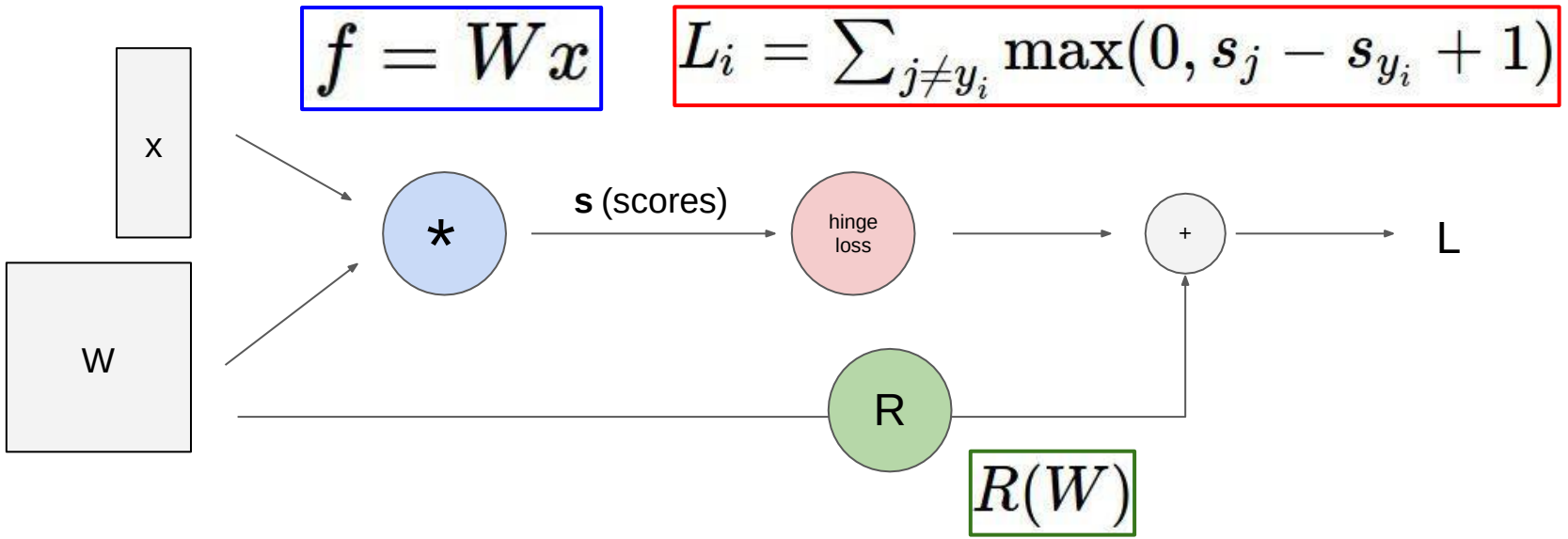
$$\nabla_W L = \nabla_W \left(\frac{1}{N} \sum_{i=1}^{N} \sum_{j \neq y_i} \max(0, W_{j,:} \cdot x + W_{y_i,:} \cdot x + 1) + \lambda \sum_k W_k^2 \right)$$

Problem: Very tedious: Lots of matrix calculus, need lots of paper

Problem: What if we want to change loss? E.g. L2 other than L1? Need to re-derive from scratch =(

Problem: Not feasible for complex models!

Better Idea: Computational graphs + Backpropagation



Credit to Stanford CS 231n

Neural Networks

Back propogation

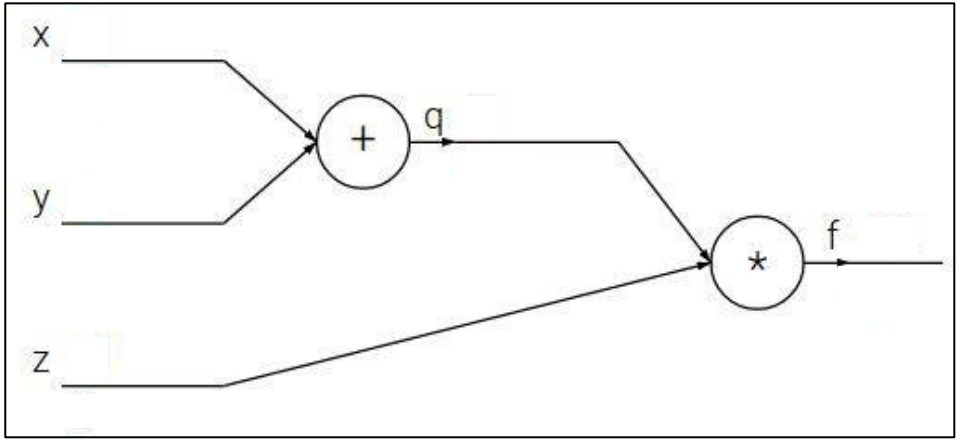
Backpropagation: a simple example

$$f(x, y, z) = (x + y)z$$

Credit to Stanford CS 231n

Backpropagation: a simple example

$$f(x, y, z) = (x + y)z$$

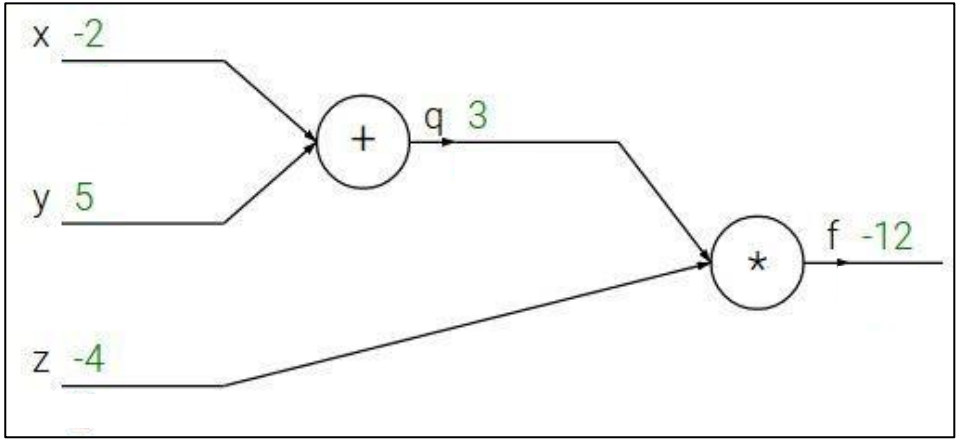


Credit to Stanford CS 231n

Backpropagation: a simple example

$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4



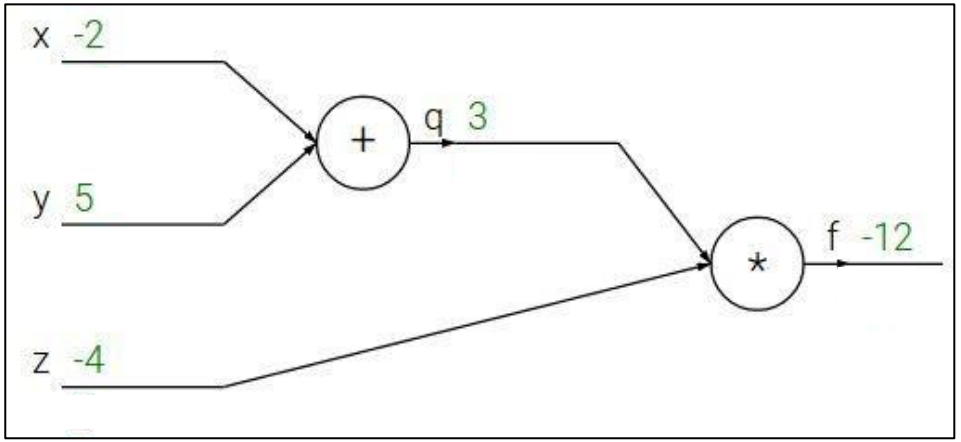
Credit to Stanford CS 231n

Backpropagation: a simple example

$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

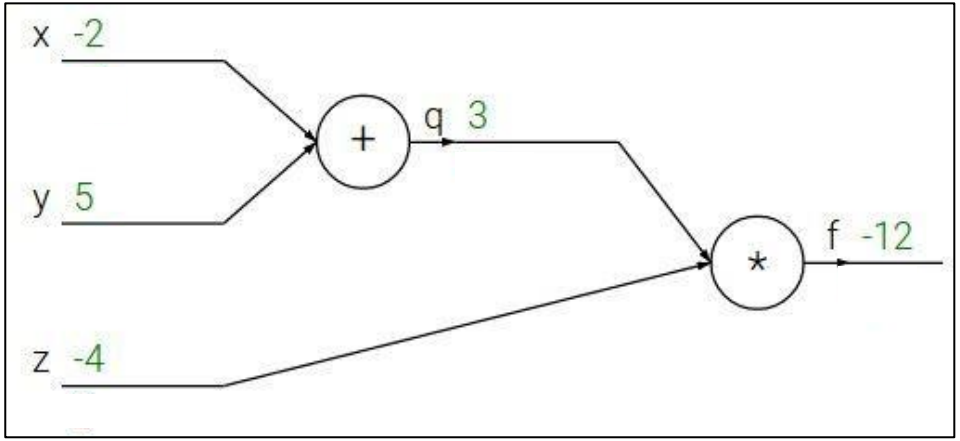


Credit to Stanford CS 231n

Backpropagation: a simple example

$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4



$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

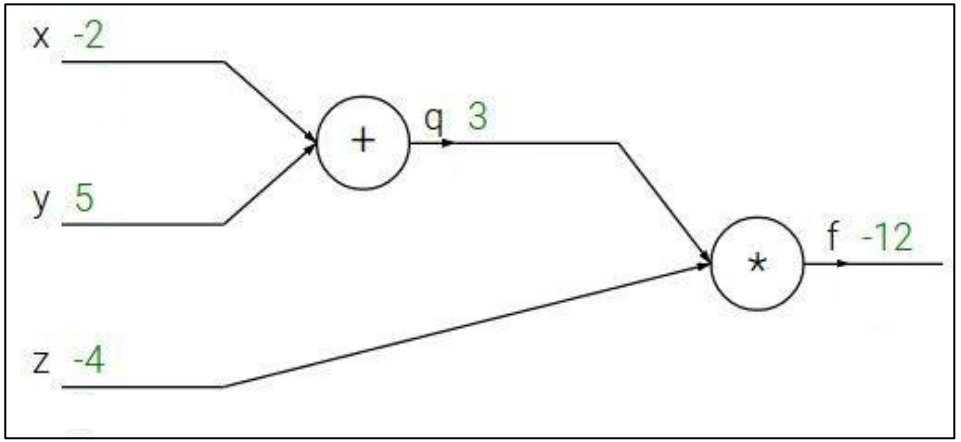
$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Credit to Stanford CS 231n

Backpropagation: a simple example

$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4



$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

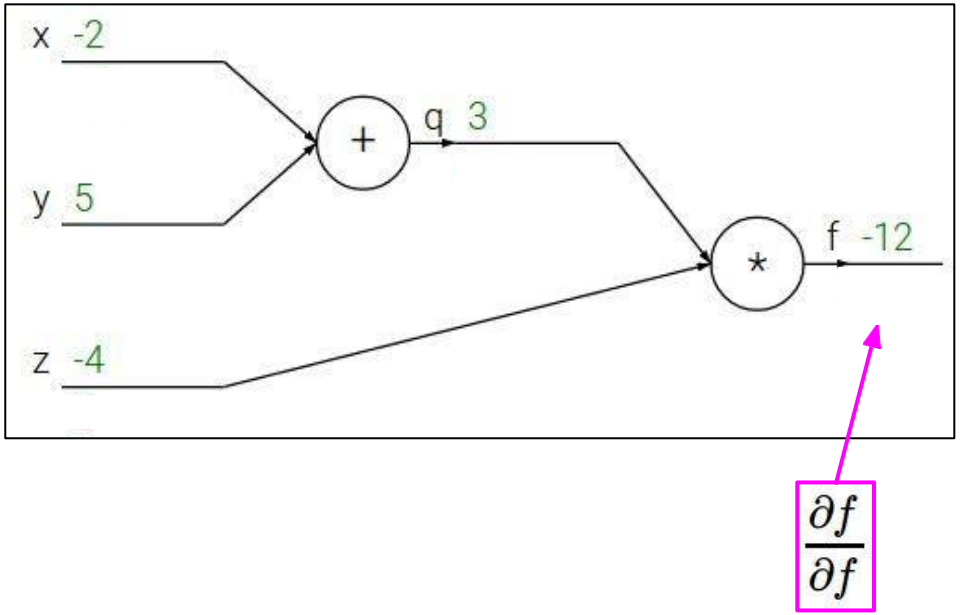
Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

Credit to Stanford CS 231n

Backpropagation: a simple example

$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4



$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

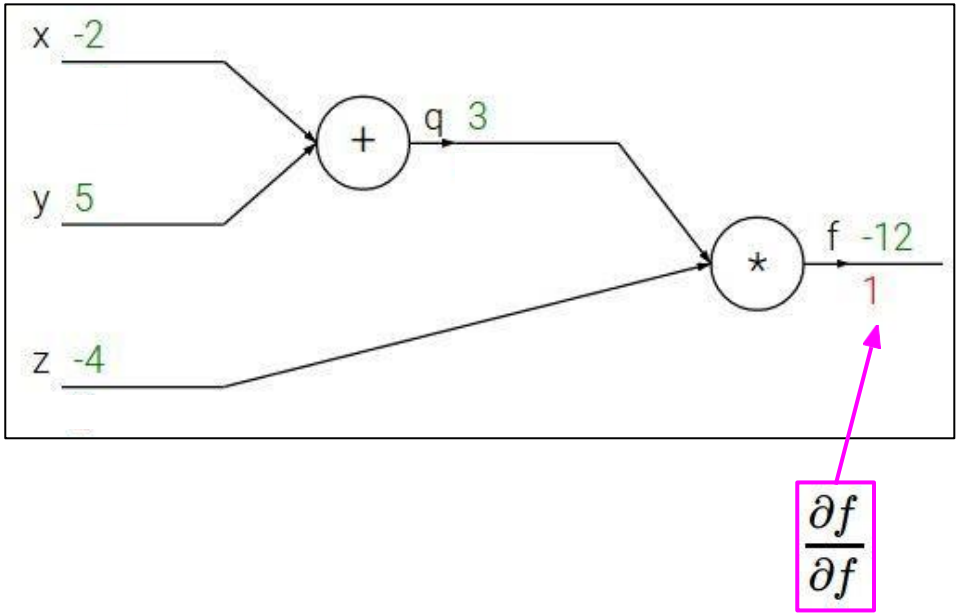
Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

Credit to Stanford CS 231n

Backpropagation: a simple example

$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4



$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

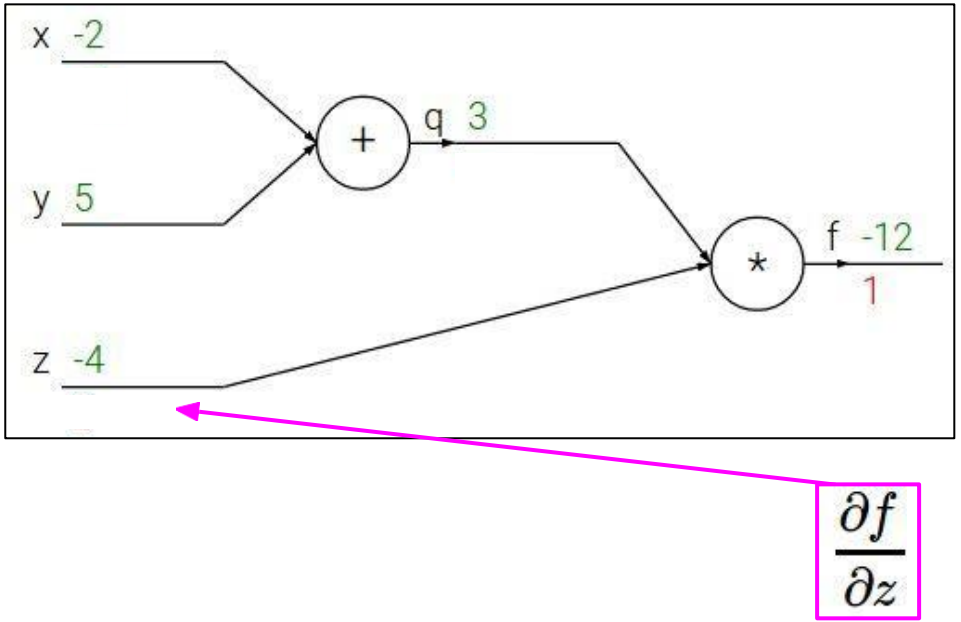
Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

Credit to Stanford CS 231n

Backpropagation: a simple example

$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4



$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

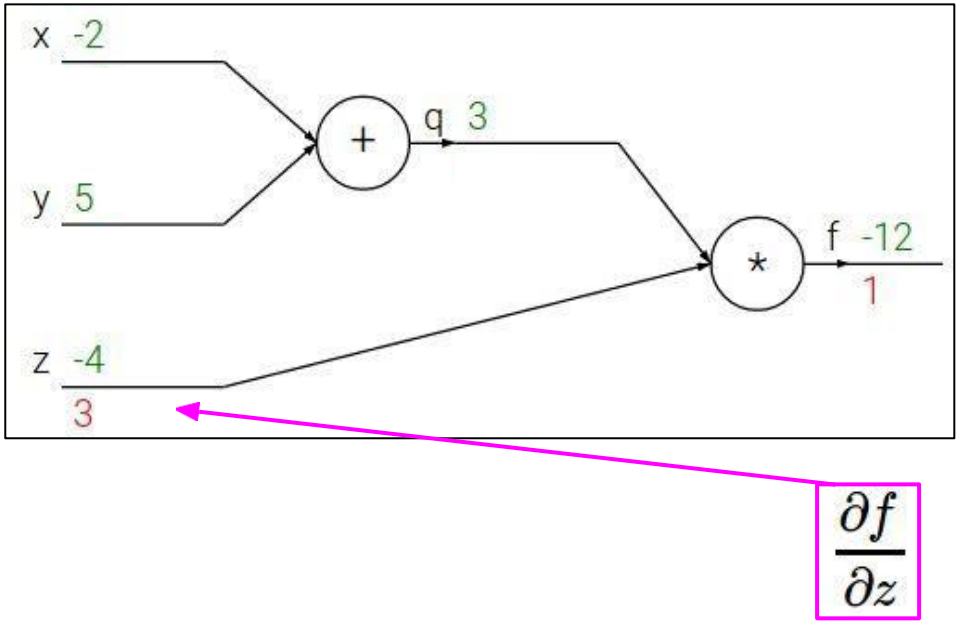
Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

Credit to Stanford CS 231n

Backpropagation: a simple example

$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4



$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

Credit to Stanford CS 231n

Backpropagation: a simple example

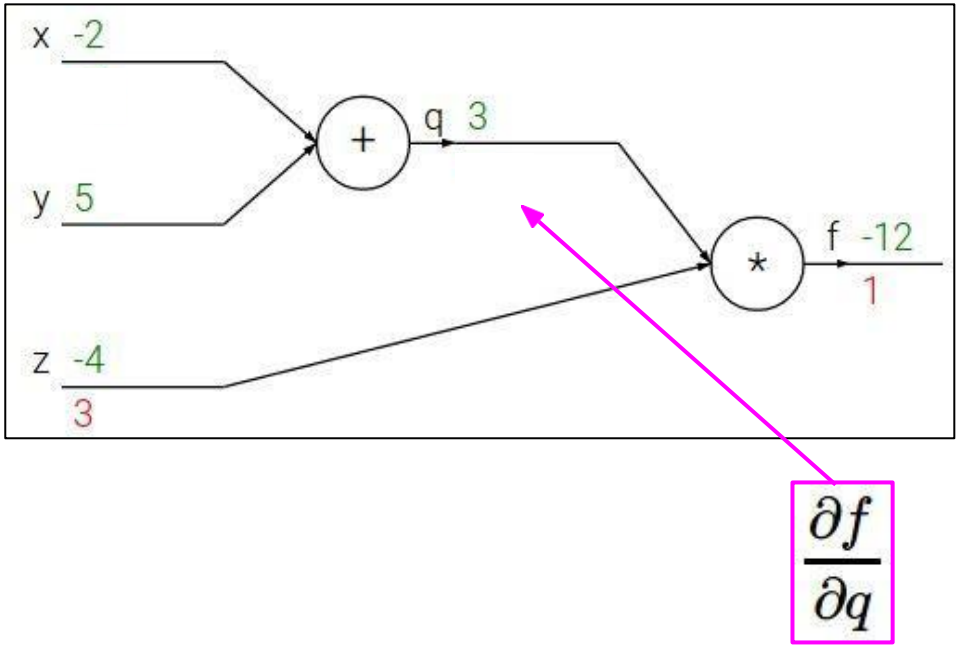
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Credit to Stanford CS 231n

Backpropagation: a simple example

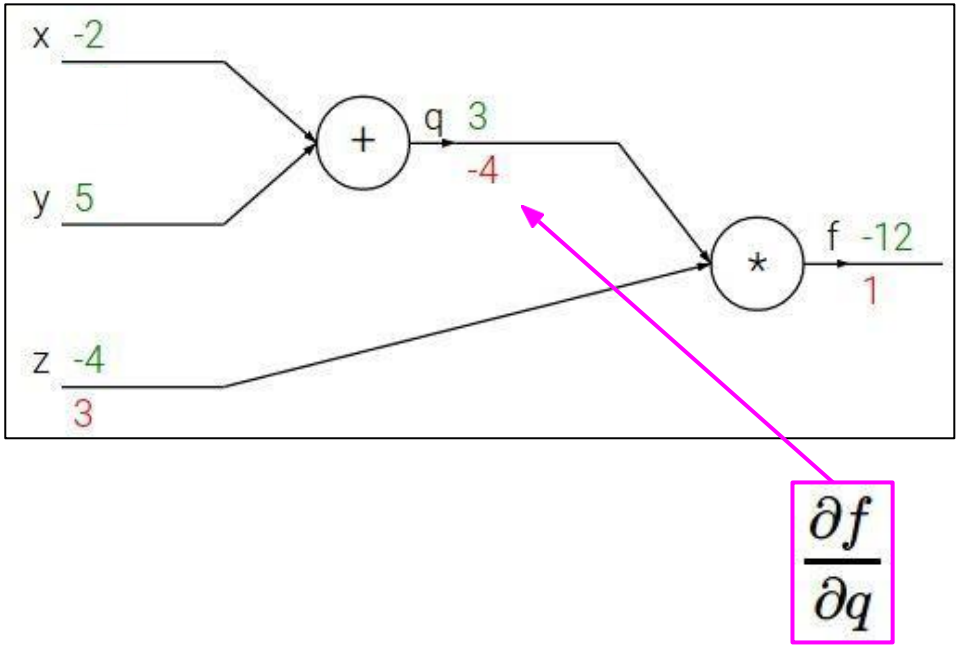
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

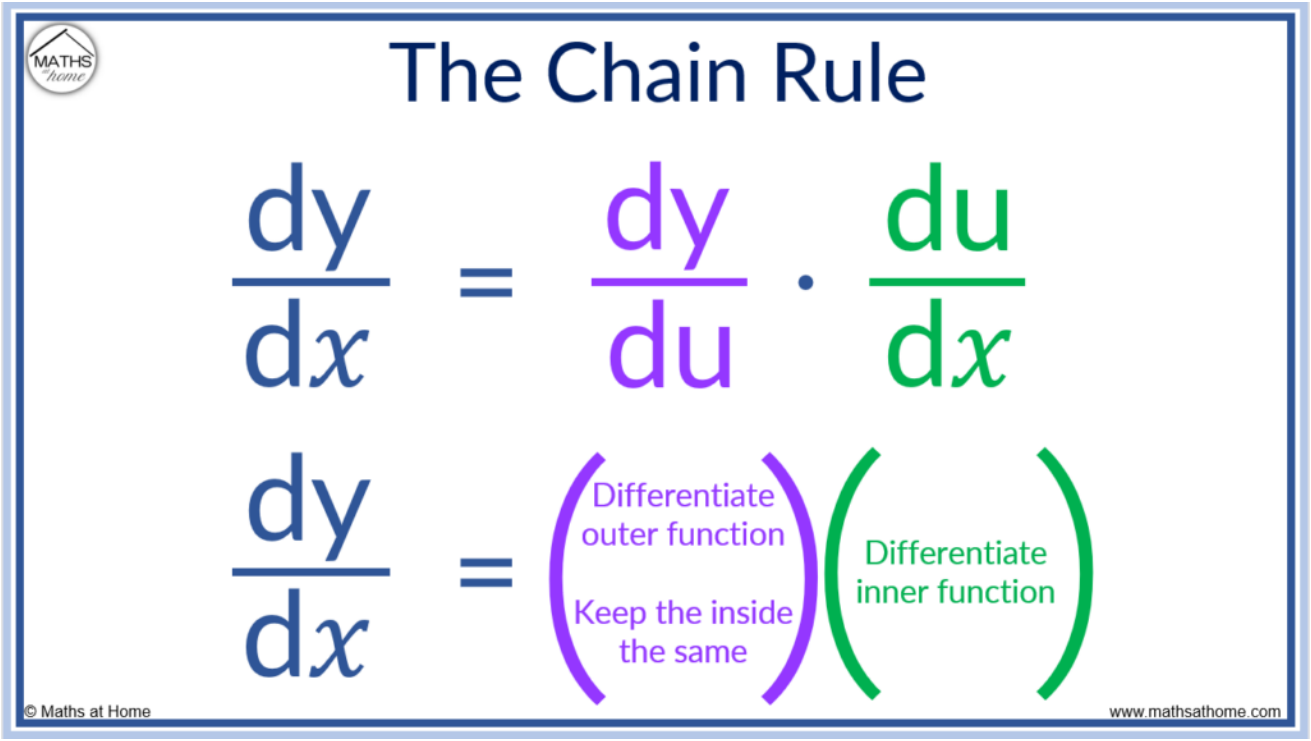
$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Credit to Stanford CS 231n

Chain Rule Revisited



5.1.2 Differentiation Rules

In the following, we briefly state basic differentiation rules, where we denote the derivative of f by f'.

$$\text{Product rule:} \quad (f(x)g(x))' = f'(x)g(x) + f(x)g'(x) \tag{5.29}$$

$$\text{Quotient rule:} \quad \left(\frac{f(x)}{g(x)}\right)' = \frac{f'(x)g(x) - f(x)g'(x)}{(g(x))^2} \tag{5.30}$$

$$\text{Sum rule:} \quad (f(x) + g(x))' = f'(x) + g'(x) \tag{5.31}$$

$$\text{Chain rule:} \quad \big(g(f(x))\big)' = (g \circ f)'(x) = g'(f(x))f'(x) \tag{5.32}$$

Here, $g \circ f$ denotes function composition $x \mapsto f(x) \mapsto g(f(x))$.

Example 5.5 (Chain Rule)
Let us compute the derivative of the function $h(x) = (2x + 1)^4$ using the chain rule. With

$$h(x) = (2x + 1)^4 = g(f(x)), \tag{5.33}$$

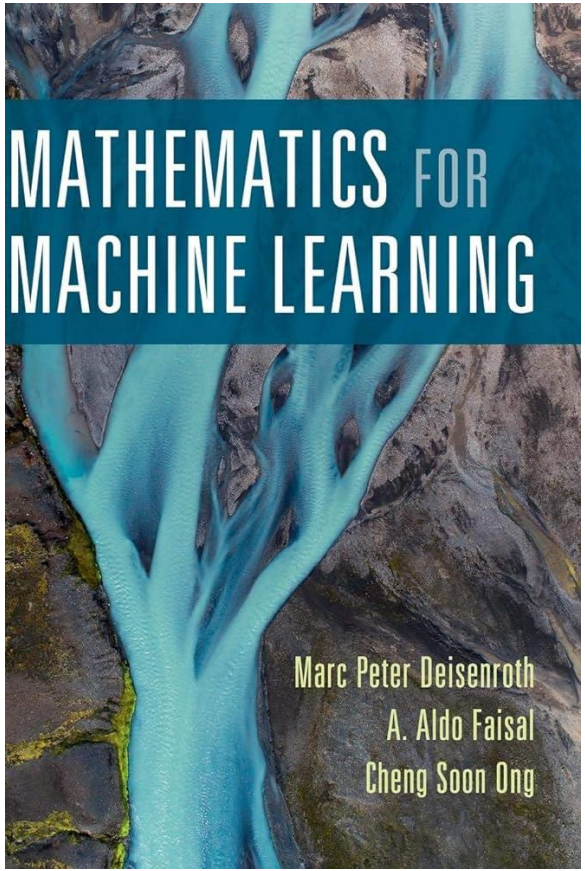
$$f(x) = 2x + 1, \tag{5.34}$$

$$g(f) = f^4, \tag{5.35}$$

we obtain the derivatives of f and g as

$$f'(x) = 2, \tag{5.36}$$

$$g'(f) = 4f^3, \tag{5.37}$$



5.2 Partial Differentiation and Gradients

Differentiation as discussed in Section 5.1 applies to functions f of a scalar variable $x \in \mathbb{R}$. In the following, we consider the general case where the function f depends on one or more variables $\boldsymbol{x} \in \mathbb{R}^n$, e.g., $f(\boldsymbol{x}) = f(x_1, x_2)$. The generalization of the derivative to functions of several variables is the *gradient*.

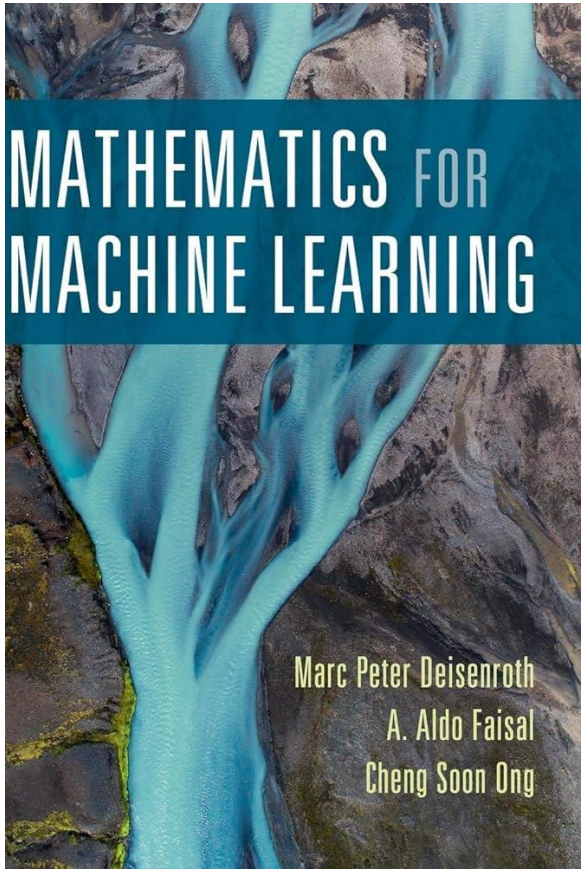
We find the gradient of the function f with respect to $\boldsymbol{x}$ by *varying one variable at a time* and keeping the others constant. The gradient is then the collection of these *partial derivatives*.

Definition 5.5 (Partial Derivative)**.** For a function $f : \mathbb{R}^n \to \mathbb{R}$, $\boldsymbol{x} \mapsto f(\boldsymbol{x})$, $\boldsymbol{x} \in \mathbb{R}^n$ of n variables $x_1, \ldots, x_n$ we define the *partial derivatives* as

$$\begin{aligned} \frac{\partial f}{\partial x_1} &= \lim_{h \to 0} \frac{f(x_1 + h, x_2, \ldots, x_n) - f(\boldsymbol{x})}{h} \\ &\vdots \\ \frac{\partial f}{\partial x_n} &= \lim_{h \to 0} \frac{f(x_1, \ldots, x_{n-1}, x_n + h) - f(\boldsymbol{x})}{h} \end{aligned} \tag{5.39}$$

and collect them in the row vector

$$\nabla_{\boldsymbol{x}} f = \operatorname{grad} f = \frac{\mathrm{d}f}{\mathrm{d}\boldsymbol{x}} = \left[\frac{\partial f(\boldsymbol{x})}{\partial x_1} \quad \frac{\partial f(\boldsymbol{x})}{\partial x_2} \quad \cdots \quad \frac{\partial f(\boldsymbol{x})}{\partial x_n} \right] \in \mathbb{R}^{1 \times n}, \tag{5.40}$$



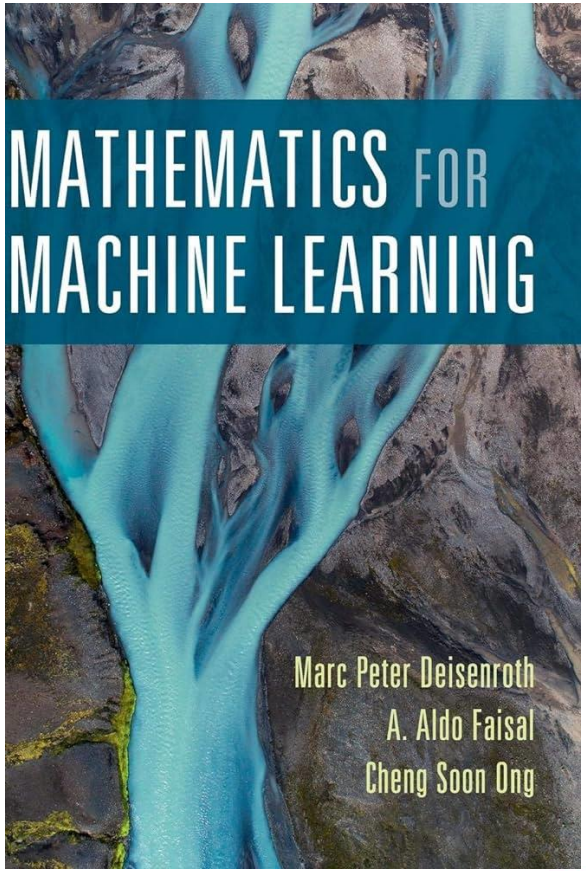
Example 5.7 (Gradient)
For $f(x_1, x_2) = x_1^2 x_2 + x_1 x_2^3 \in \mathbb{R}$, the partial derivatives (i.e., the derivatives of f with respect to x_1 and x_2) are

$$\frac{\partial f(x_1, x_2)}{\partial x_1} = 2x_1 x_2 + x_2^3 \tag{5.43}$$

$$\frac{\partial f(x_1, x_2)}{\partial x_2} = x_1^2 + 3x_1 x_2^2 \tag{5.44}$$

and the gradient is then

$$\frac{\mathrm{d}f}{\mathrm{d}\boldsymbol{x}} = \begin{bmatrix} \frac{\partial f(x_1, x_2)}{\partial x_1} & \frac{\partial f(x_1, x_2)}{\partial x_2} \end{bmatrix} = \begin{bmatrix} 2x_1 x_2 + x_2^3 & x_1^2 + 3x_1 x_2^2 \end{bmatrix} \in \mathbb{R}^{1\times 2} . \tag{5.45}$$



5.2.2 Chain Rule

Consider a function $f : \mathbb{R}^2 \to \mathbb{R}$ of two variables x_1, x_2. Furthermore, $x_1(t)$ and $x_2(t)$ are themselves functions of t. To compute the gradient of f with respect to t, we need to apply the chain rule (5.48) for multivariate functions as

$$\frac{\mathrm{d}f}{\mathrm{d}t} = \begin{bmatrix} \frac{\partial f}{\partial x_1} & \frac{\partial f}{\partial x_2} \end{bmatrix} \begin{bmatrix} \frac{\partial x_1(t)}{\partial t} \\ \frac{\partial x_2(t)}{\partial t} \end{bmatrix} = \frac{\partial f}{\partial x_1}\frac{\partial x_1}{\partial t} + \frac{\partial f}{\partial x_2}\frac{\partial x_2}{\partial t}, \tag{5.49}$$

where d denotes the gradient and ∂ partial derivatives.

Example 5.8

Consider $f(x_1, x_2) = x_1^2 + 2x_2$, where $x_1 = \sin t$ and $x_2 = \cos t$, then

$$\frac{\mathrm{d}f}{\mathrm{d}t} = \frac{\partial f}{\partial x_1}\frac{\partial x_1}{\partial t} + \frac{\partial f}{\partial x_2}\frac{\partial x_2}{\partial t} \tag{5.50a}$$

$$= 2\sin t \frac{\partial \sin t}{\partial t} + 2\frac{\partial \cos t}{\partial t} \tag{5.50b}$$

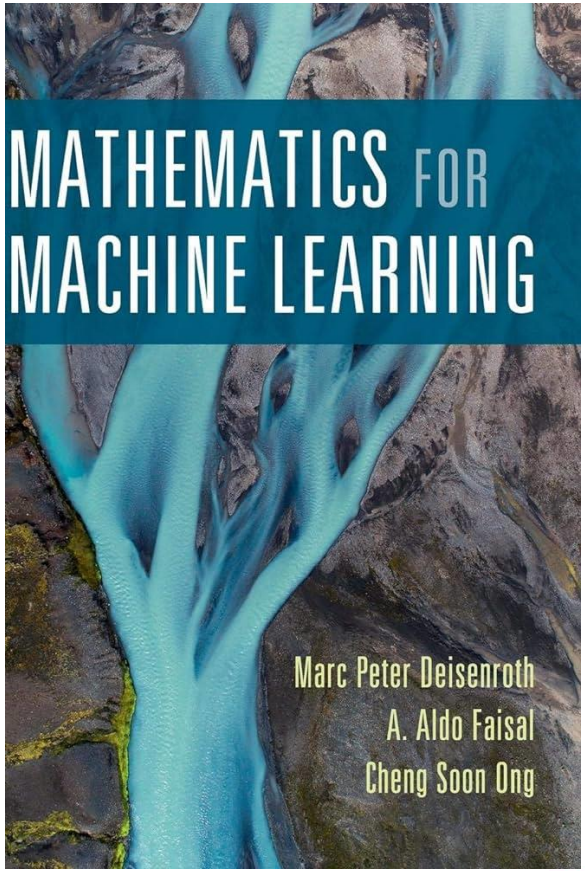
$$= 2\sin t \cos t - 2\sin t = 2\sin t(\cos t - 1) \tag{5.50c}$$

is the corresponding derivative of f with respect to t.

If $f(x_1, x_2)$ is a function of x_1 and x_2, where $x_1(s, t)$ and $x_2(s, t)$ are themselves functions of two variables s and t, the chain rule yields the partial derivatives

$$\frac{\partial f}{\partial s} = \frac{\partial f}{\partial x_1}\frac{\partial x_1}{\partial s} + \frac{\partial f}{\partial x_2}\frac{\partial x_2}{\partial s}, \tag{5.51}$$

$$\frac{\partial f}{\partial t} = \frac{\partial f}{\partial x_1}\frac{\partial x_1}{\partial t} + \frac{\partial f}{\partial x_2}\frac{\partial x_2}{\partial t}, \tag{5.52}$$



Backpropagation: a simple example

$$f(x, y, z) = (x + y)z$$

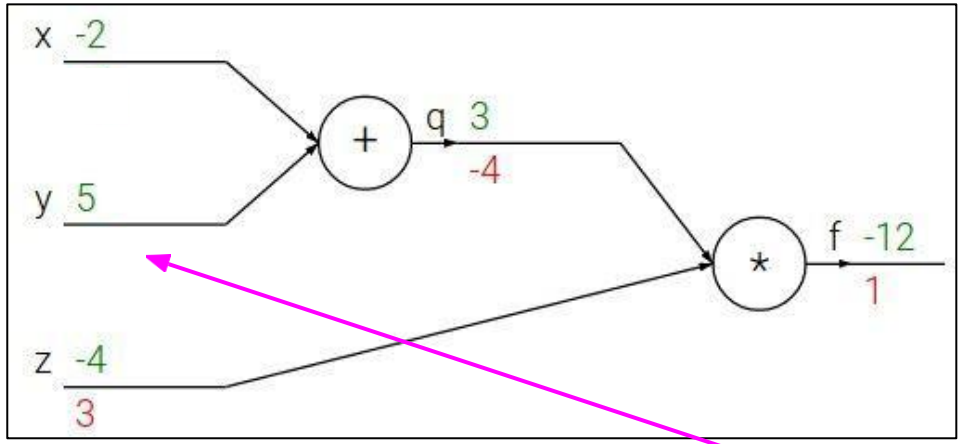
e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

Credit to Stanford CS 231n



Chain rule:

$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial y}$$

Upstream gradient

Local gradient

Backpropagation: a simple example

$$f(x, y, z) = (x + y)z$$

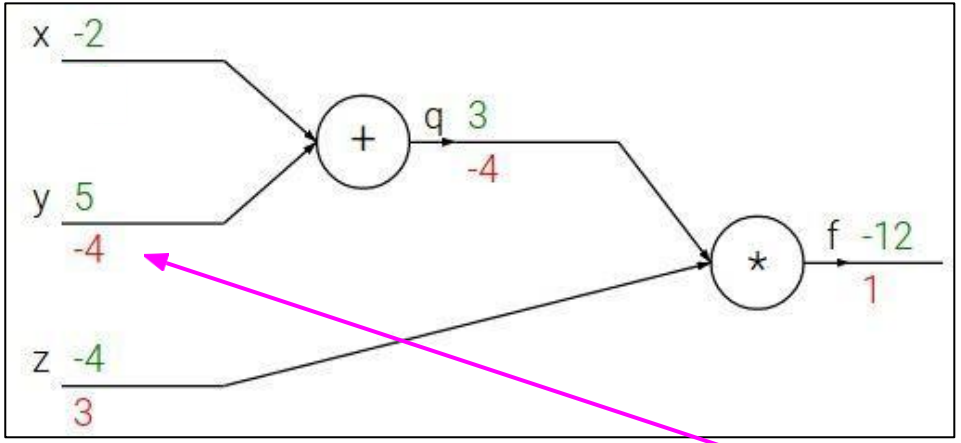
e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

Credit to Stanford CS 231n



Chain rule:

$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial y}$$

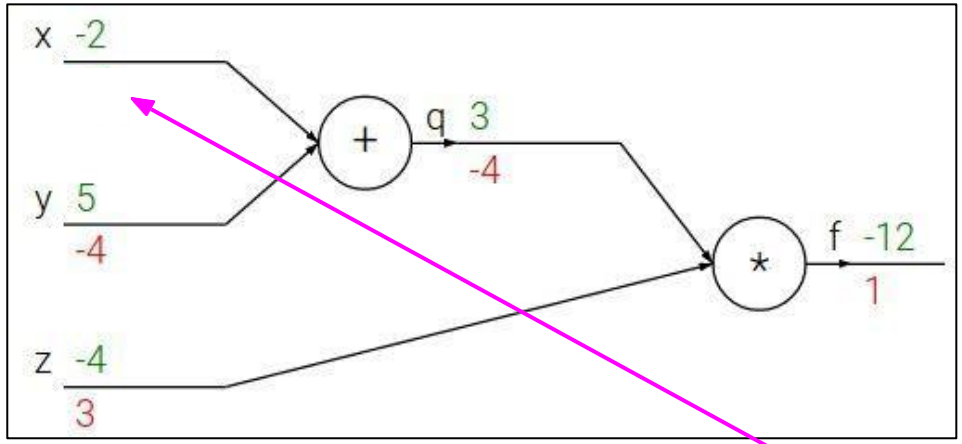
Upstream gradient

Local gradient

Backpropagation: a simple example

$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4



$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

Chain rule:

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial x}$$

Upstream gradient

Local gradient

Credit to Stanford CS 231n

Backpropagation: a simple example

$$f(x, y, z) = (x + y)z$$

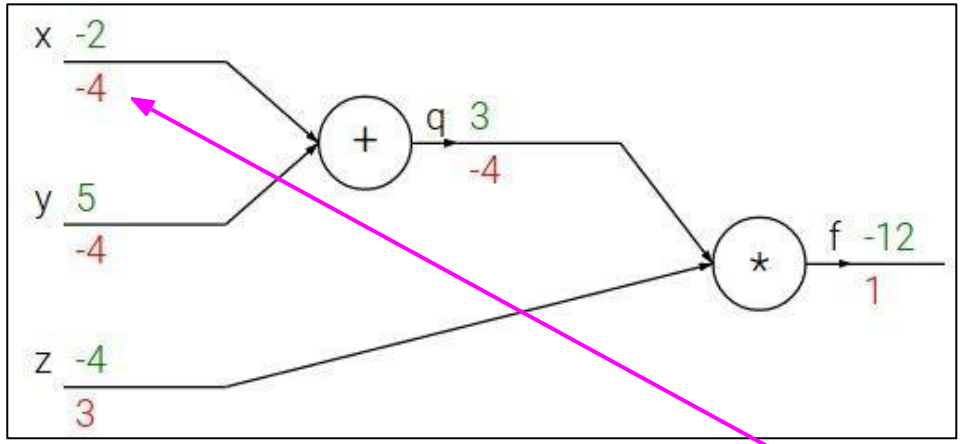
e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

Credit to Stanford CS 231n

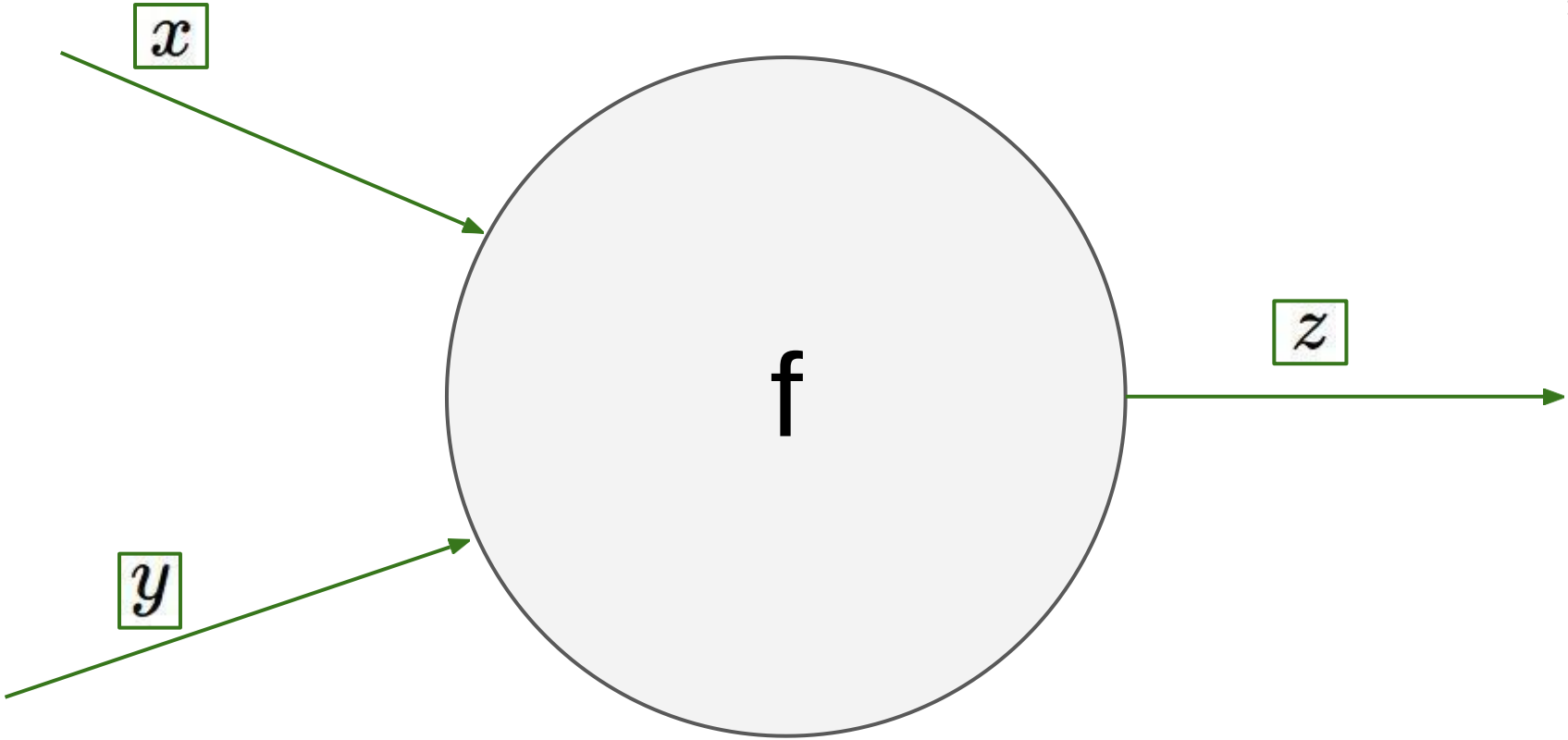


Chain rule:

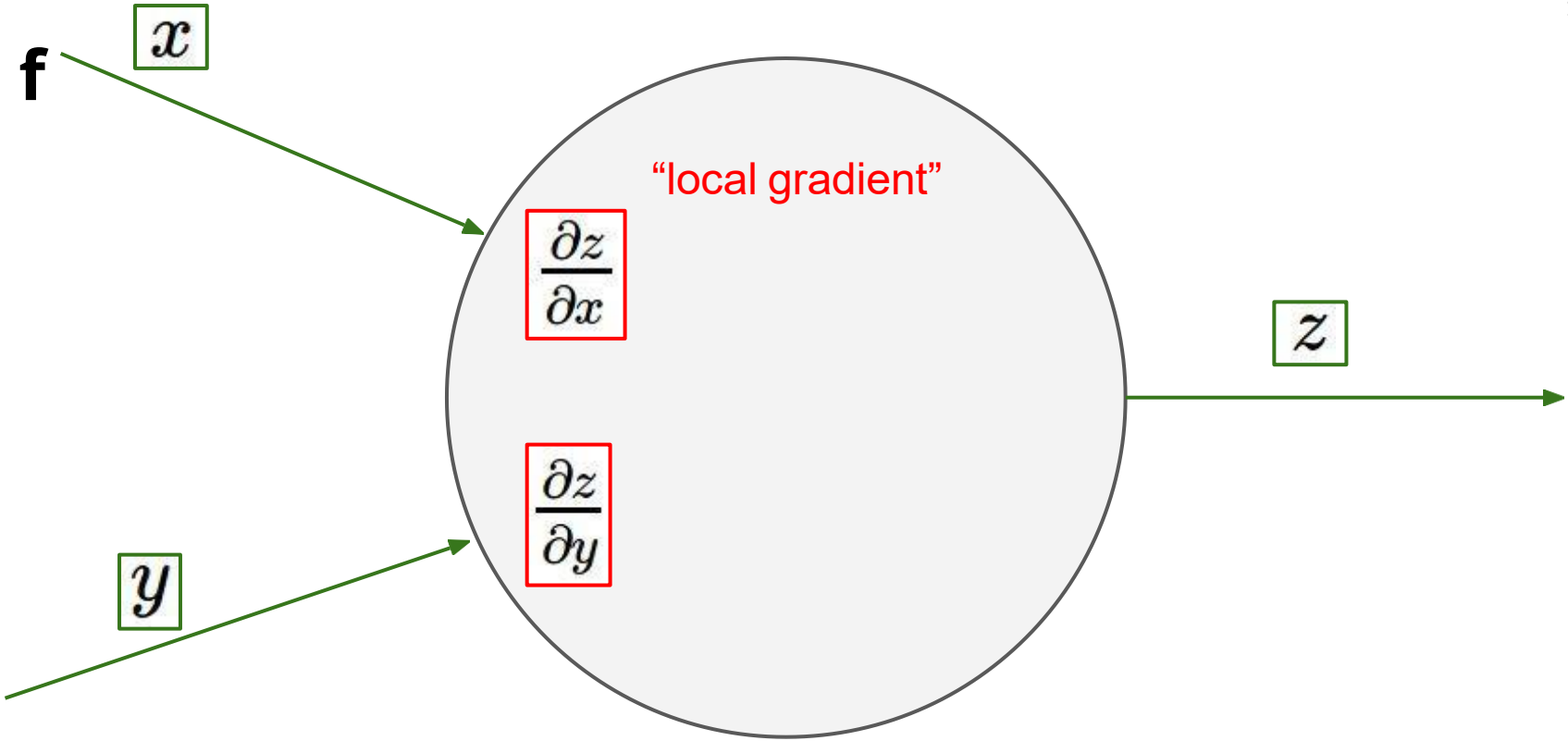
$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial x}$$

Upstream gradient

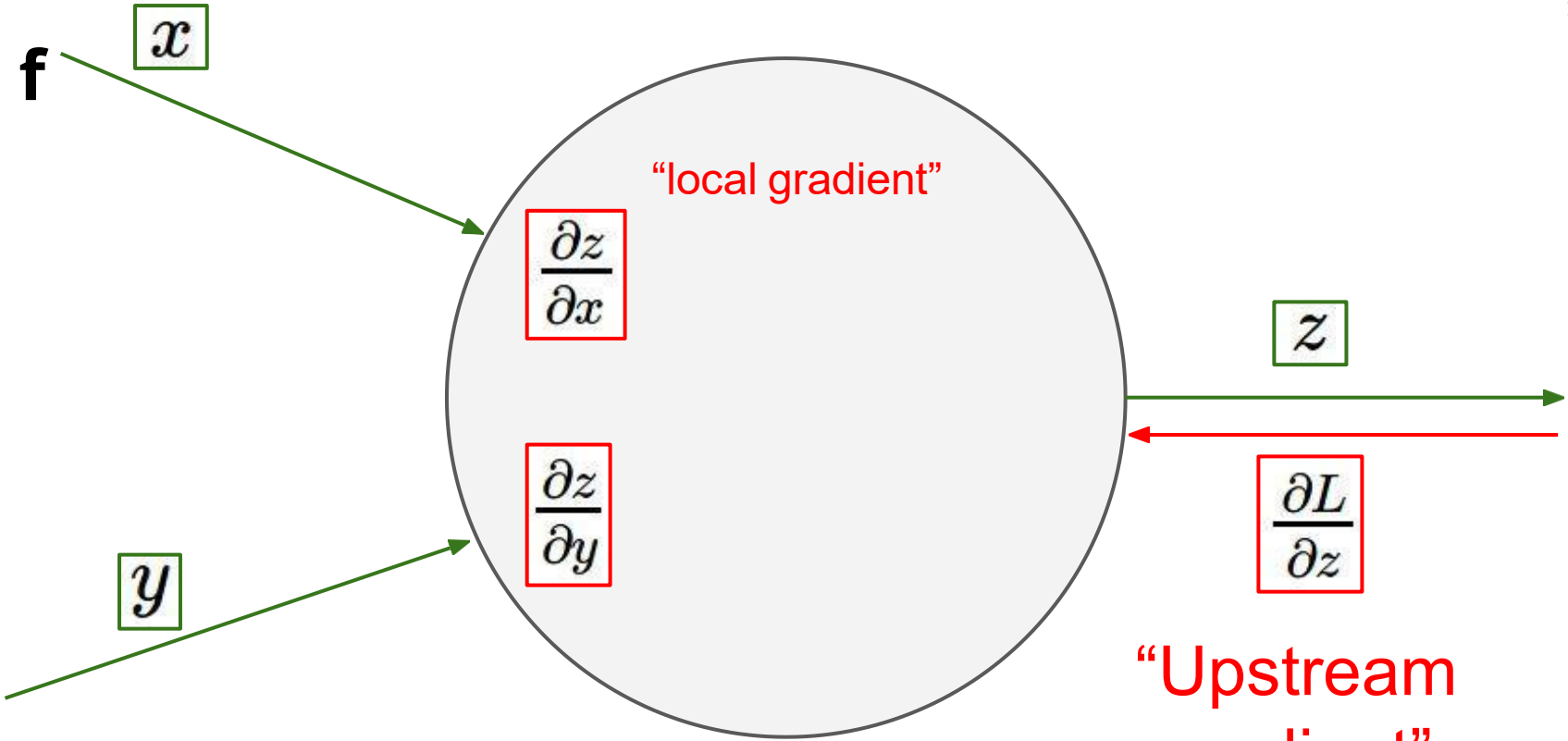
Local gradient



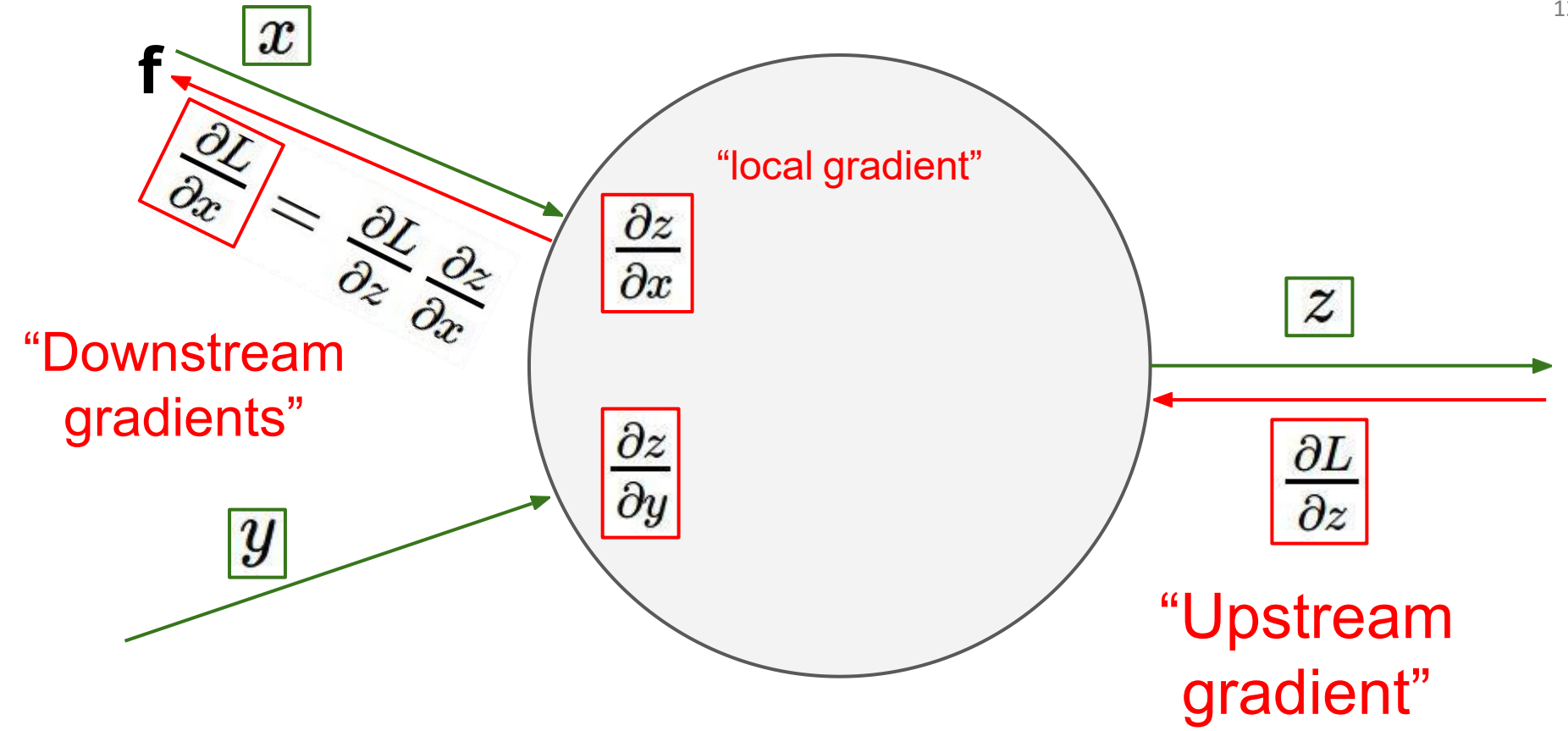
Credit to Stanford CS 231n



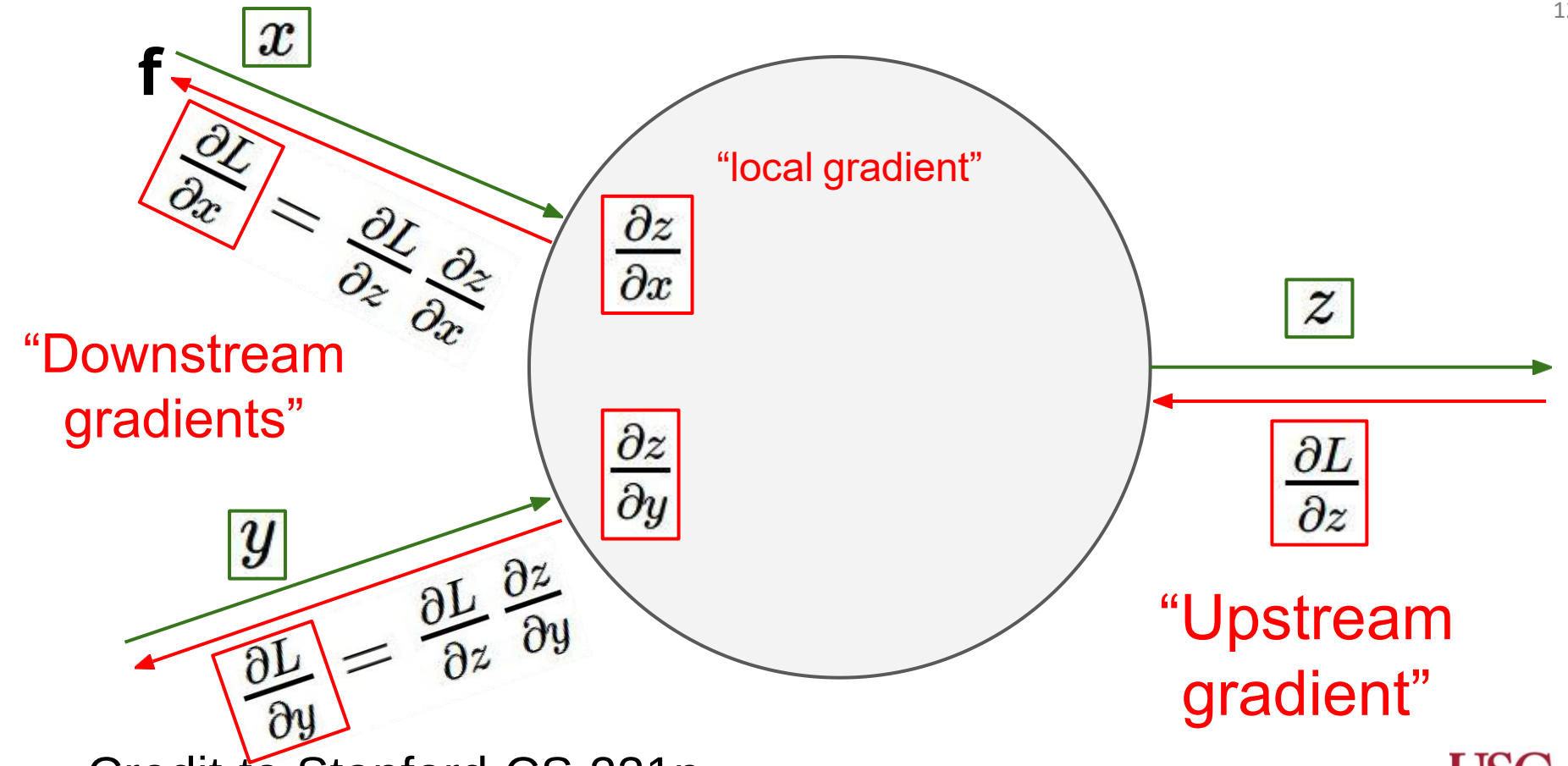
Credit to Stanford CS 231n



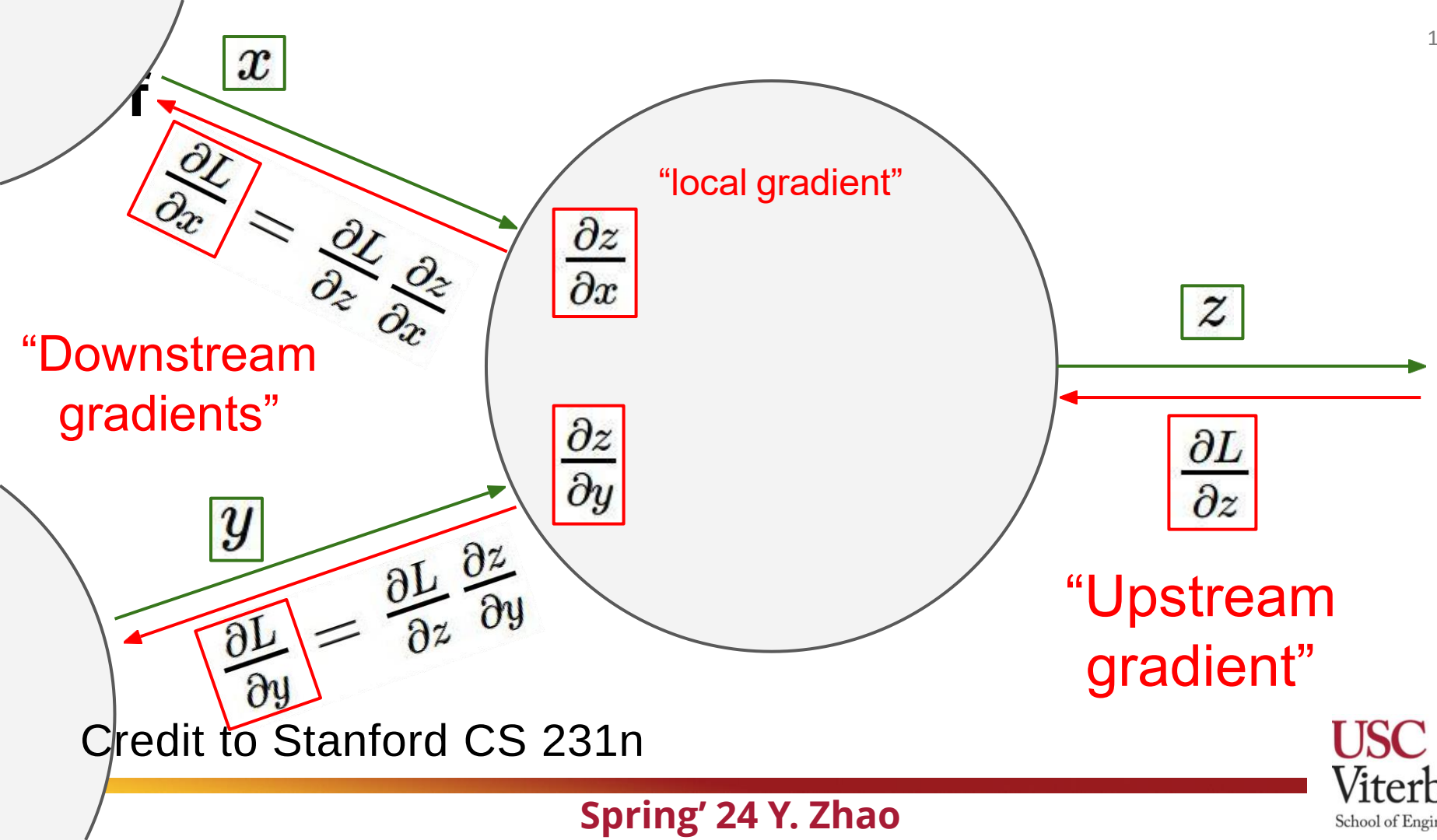
Credit to Stanford CS 231n



Credit to Stanford CS 231n



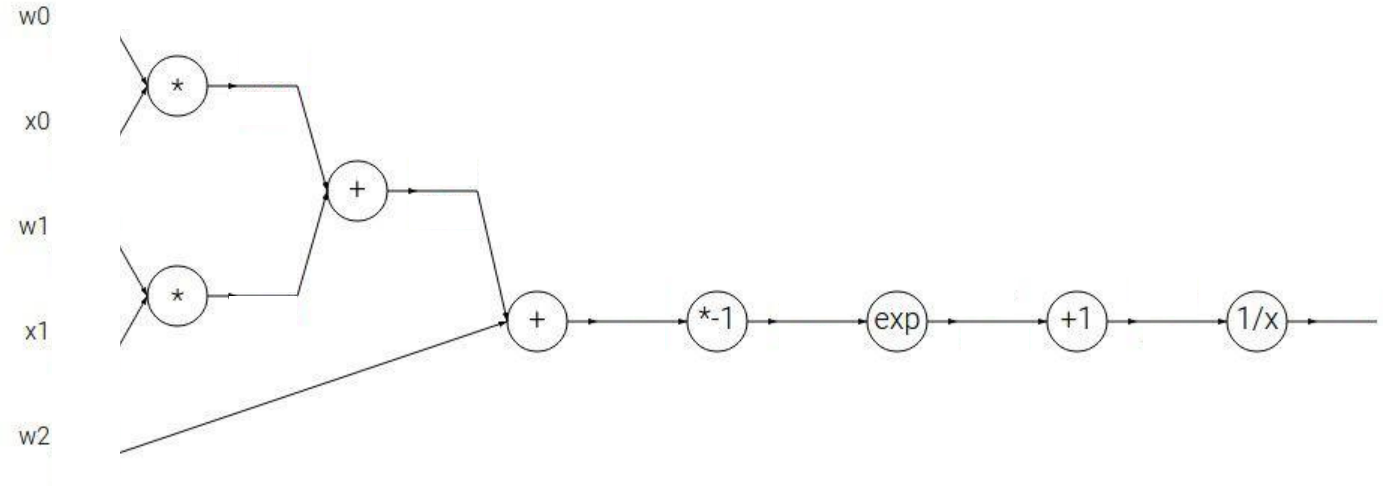
Credit to Stanford CS 231n



Credit to Stanford CS 231n

Another example:

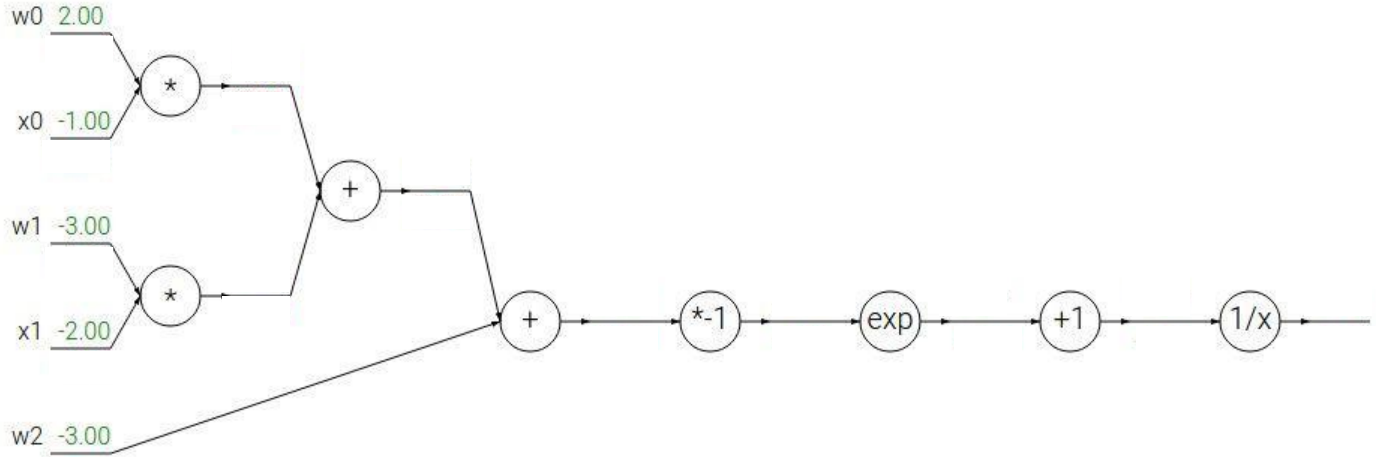
$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



Credit to Stanford CS 231n

Another example:

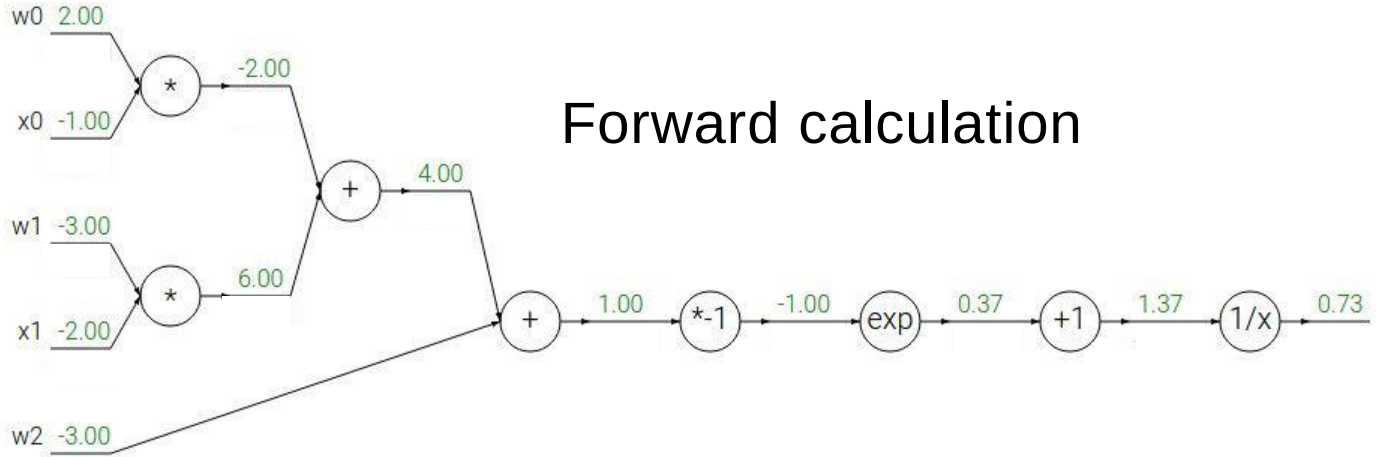
$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



Credit to Stanford CS 231n

Another example:

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$

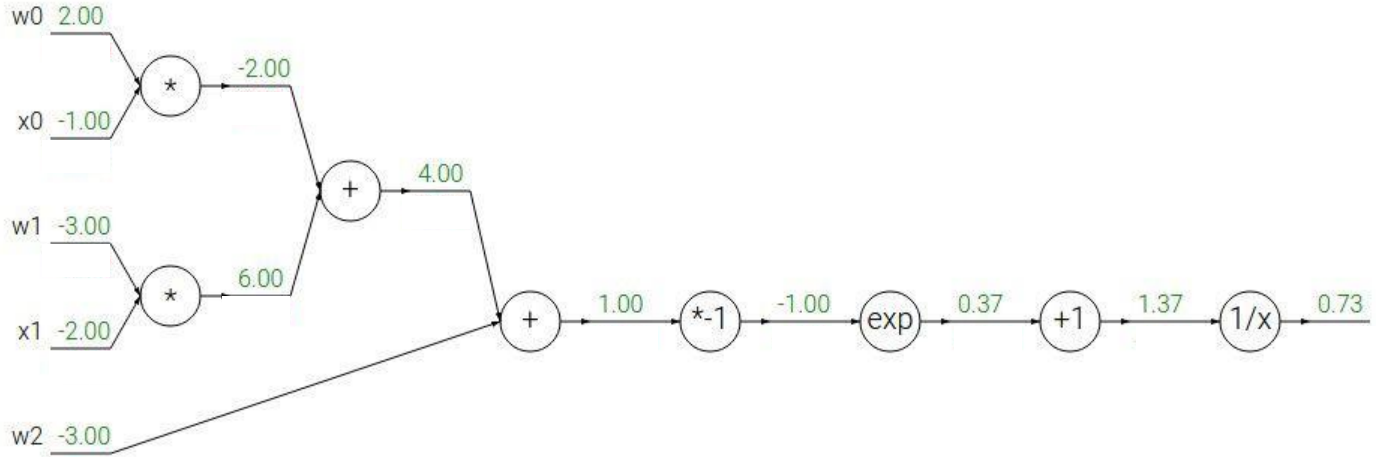


Forward calculation

Credit to Stanford CS 231n

Another example:

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



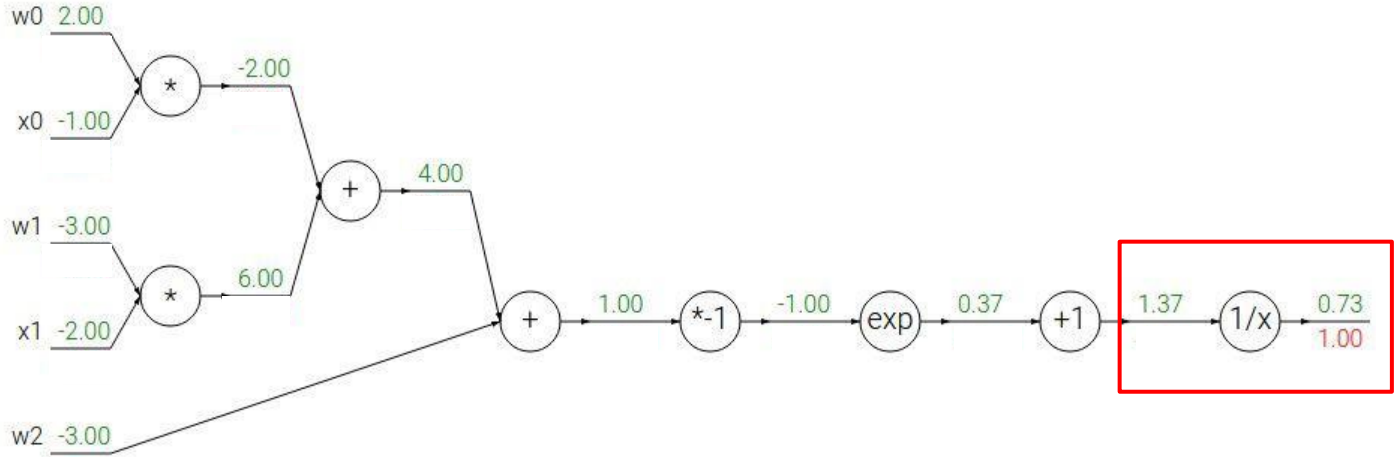
$$f(x) = e^x \rightarrow \frac{df}{dx} = e^x \qquad f(x) = \frac{1}{x} \rightarrow \frac{df}{dx} = -1/x^2$$

$$f_a(x) = ax \rightarrow \frac{df}{dx} = a \qquad f_c(x) = c + x \rightarrow \frac{df}{dx} = 1$$

Credit to Stanford CS 231n

Another example:

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



$$f(x) = e^x \rightarrow \frac{df}{dx} = e^x$$

$$f_a(x) = ax \rightarrow \frac{df}{dx} = a$$

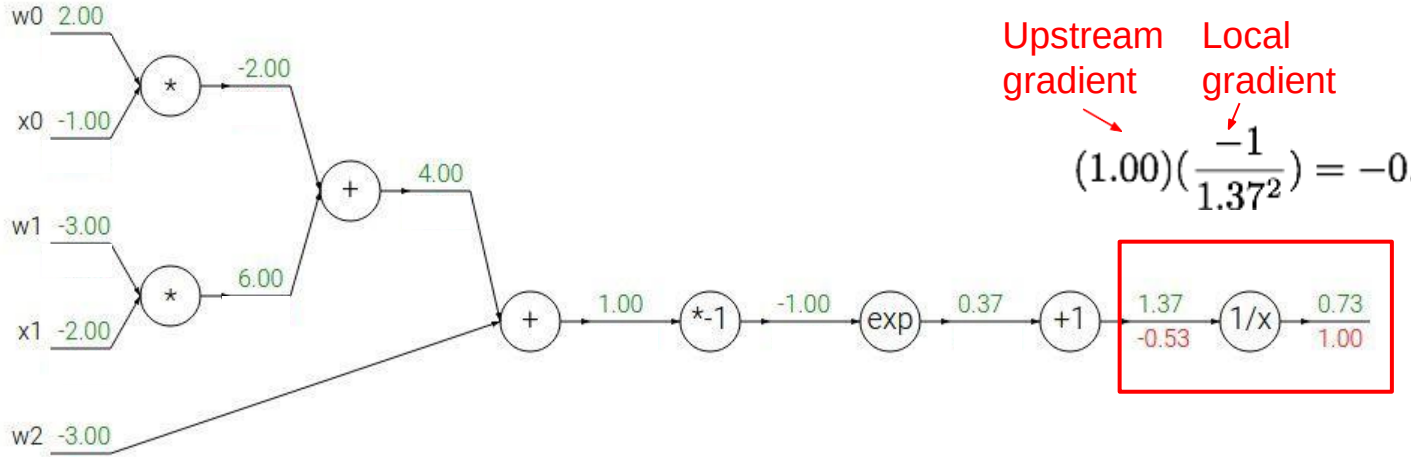
$$f(x) = \frac{1}{x} \rightarrow \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \rightarrow \frac{df}{dx} = 1$$

Credit to Stanford CS 231n

Another example:

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



Upstream gradient

Local gradient

$$(1.00)(\frac{-1}{1.37^2}) = -0.53$$

$f(x) = e^x \rightarrow \frac{df}{dx} = e^x$

$f_a(x) = ax \rightarrow \frac{df}{dx} = a$

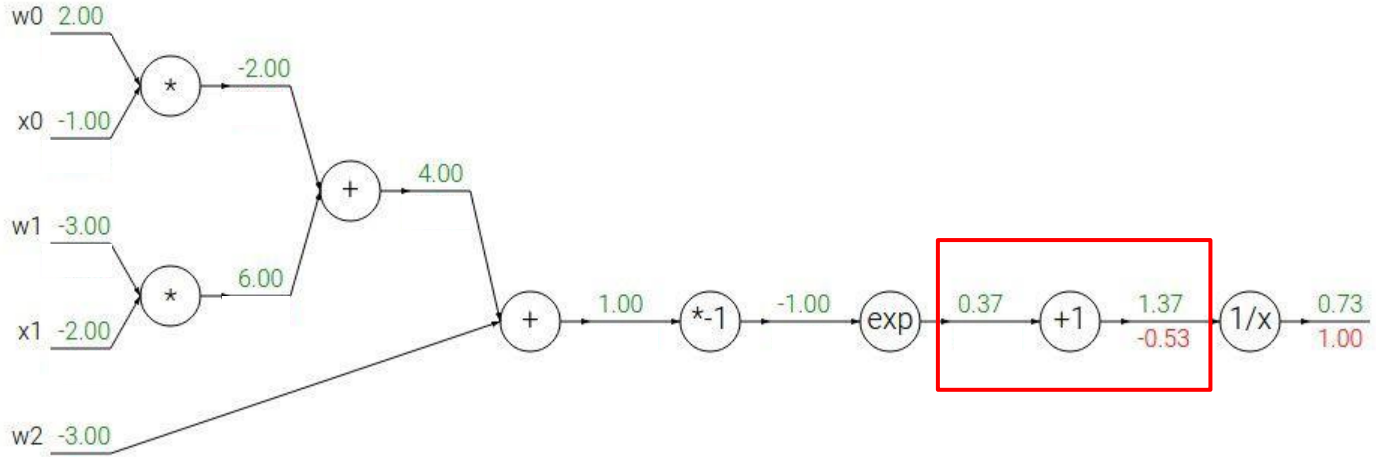
$f(x) = \frac{1}{x} \rightarrow \frac{df}{dx} = -1/x^2$

$f_c(x) = c + x \rightarrow \frac{df}{dx} = 1$

Credit to Stanford CS 231n

Another example:

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



$$f(x) = e^x \rightarrow \frac{df}{dx} = e^x$$

$$f_a(x) = ax \rightarrow \frac{df}{dx} = a$$

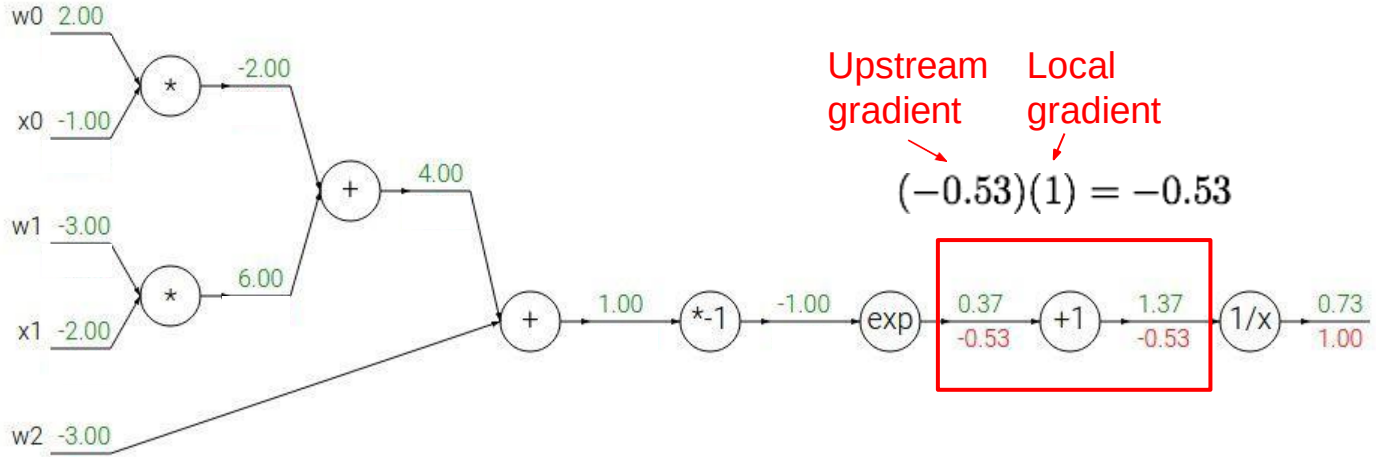
$$f(x) = \frac{1}{x} \rightarrow \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \rightarrow \frac{df}{dx} = 1$$

Credit to Stanford CS 231n

Another example:

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



$$f(x) = e^x \rightarrow \frac{df}{dx} = e^x$$

$$f_a(x) = ax \rightarrow \frac{df}{dx} = a$$

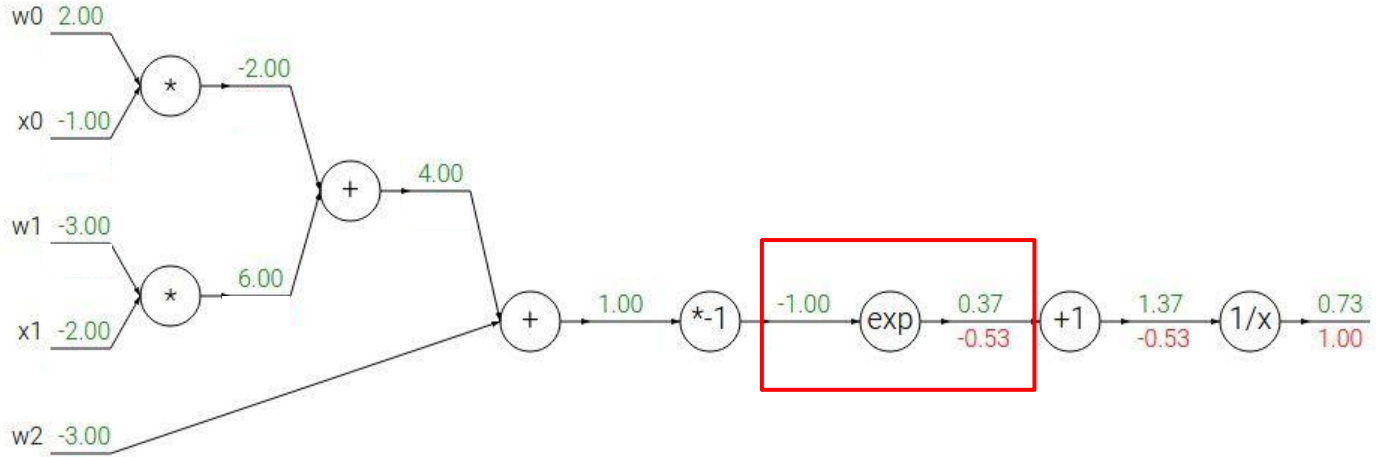
$$f(x) = \frac{1}{x} \rightarrow \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \rightarrow \frac{df}{dx} = 1$$

Credit to Stanford CS 231n

Another example:

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



$$f(x) = e^x \rightarrow \frac{df}{dx} = e^x$$

$$f_a(x) = ax \rightarrow \frac{df}{dx} = a$$

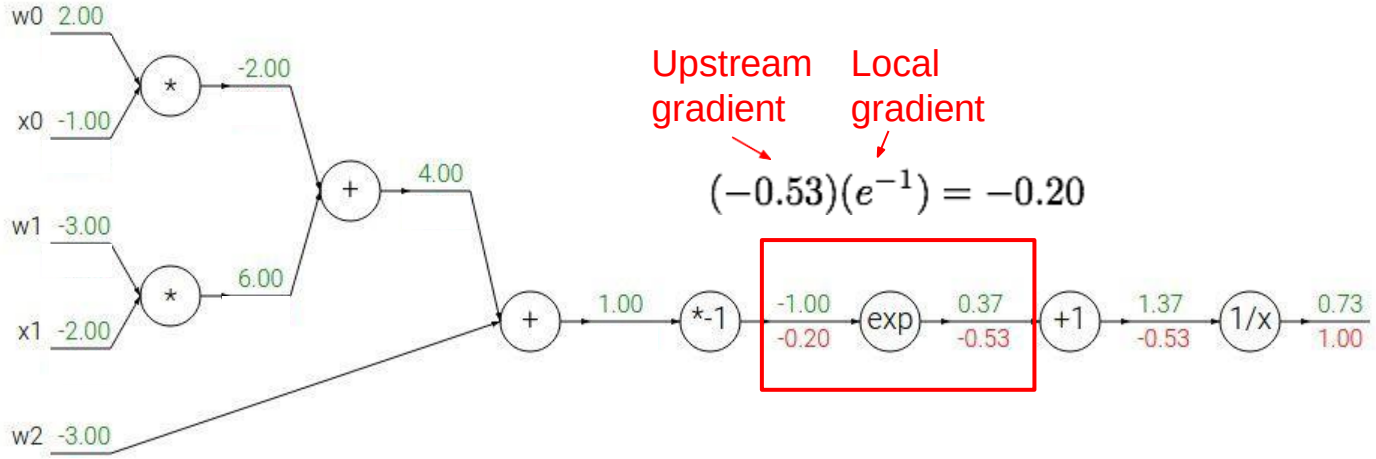
$$f(x) = \frac{1}{x} \rightarrow \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \rightarrow \frac{df}{dx} = 1$$

Credit to Stanford CS 231n

Another example:

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



$$f(x) = e^x \rightarrow \frac{df}{dx} = e^x$$

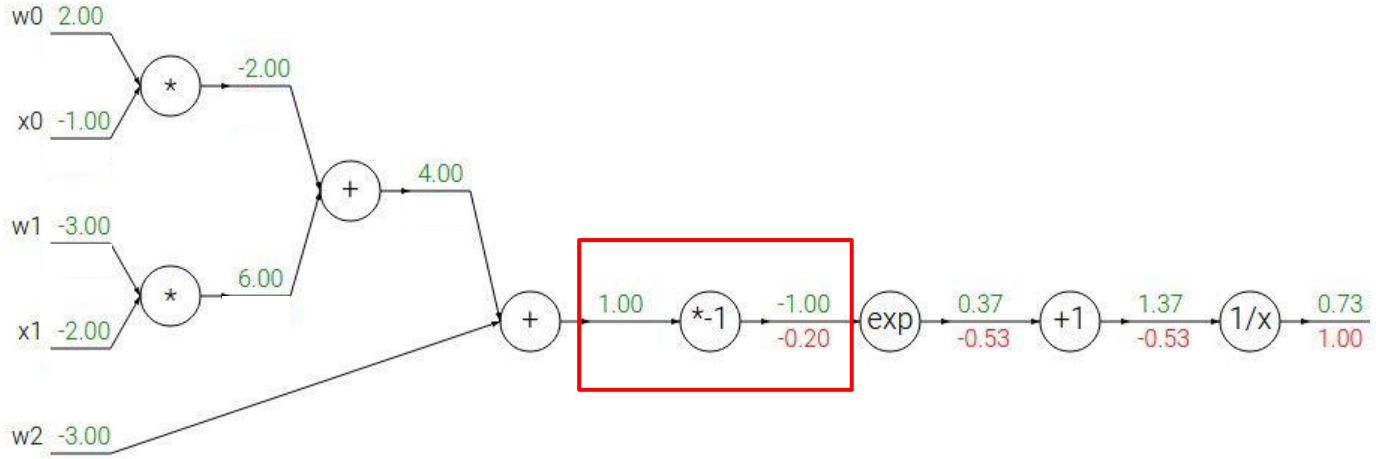
$$f_a(x) = ax \rightarrow \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \rightarrow \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \rightarrow \frac{df}{dx} = 1$$

Credit to Stanford CS 231n

Another example: $f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$



$f(x) = e^x \rightarrow \frac{df}{dx} = e^x$

$f_a(x) = ax \rightarrow \frac{df}{dx} = a$

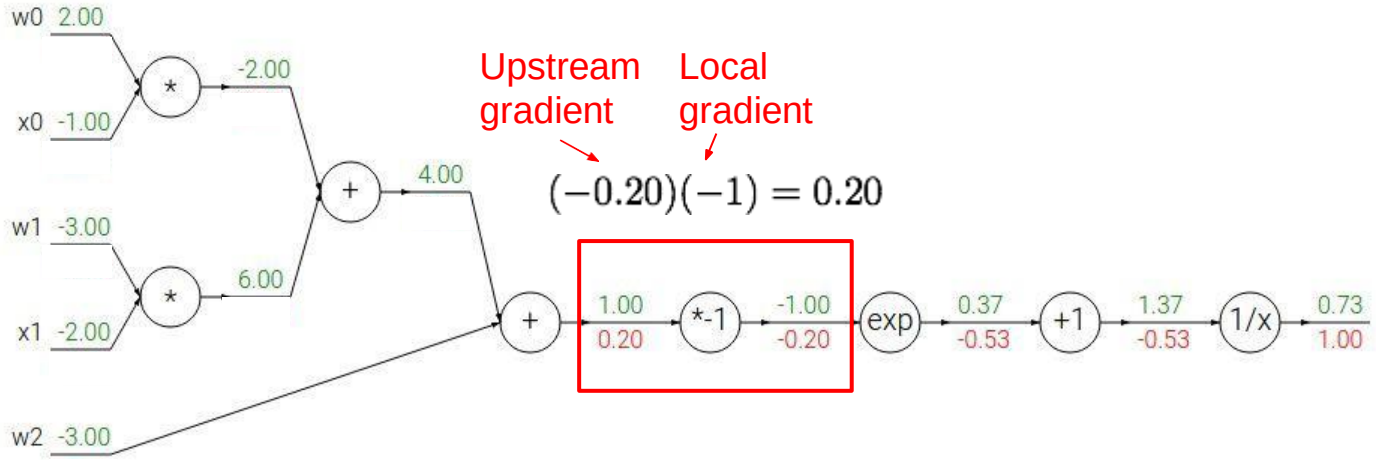
$f(x) = \frac{1}{x} \rightarrow \frac{df}{dx} = -1/x^2$

$f_c(x) = c + x \rightarrow \frac{df}{dx} = 1$

Credit to Stanford CS 231n

Another example:

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

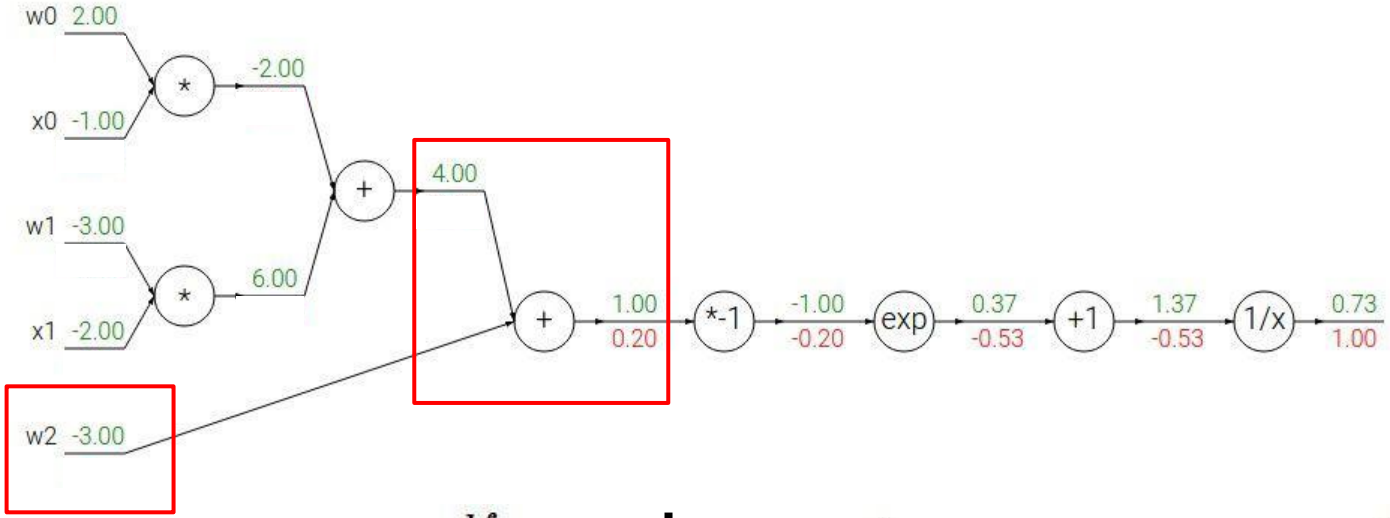
$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

Credit to Stanford CS 231n

Another example: $f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$



$f(x) = e^x \rightarrow \frac{df}{dx} = e^x$

$f_a(x) = ax \rightarrow \frac{df}{dx} = a$

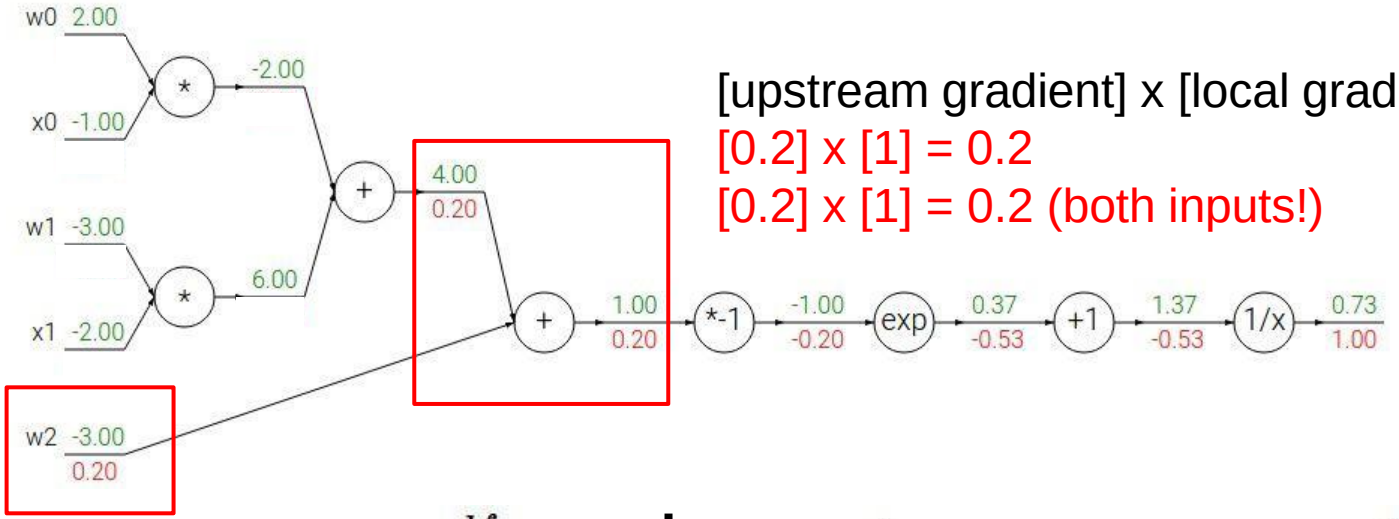
$f(x) = \frac{1}{x} \rightarrow \frac{df}{dx} = -1/x^2$

$f_c(x) = c + x \rightarrow \frac{df}{dx} = 1$

Credit to Stanford CS 231n

Another example:

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



[upstream gradient] x [local gradient]

[0.2] x [1] = 0.2

[0.2] x [1] = 0.2 (both inputs!)

$f(x) = e^x \rightarrow \frac{df}{dx} = e^x$

$f_a(x) = ax \rightarrow \frac{df}{dx} = a$

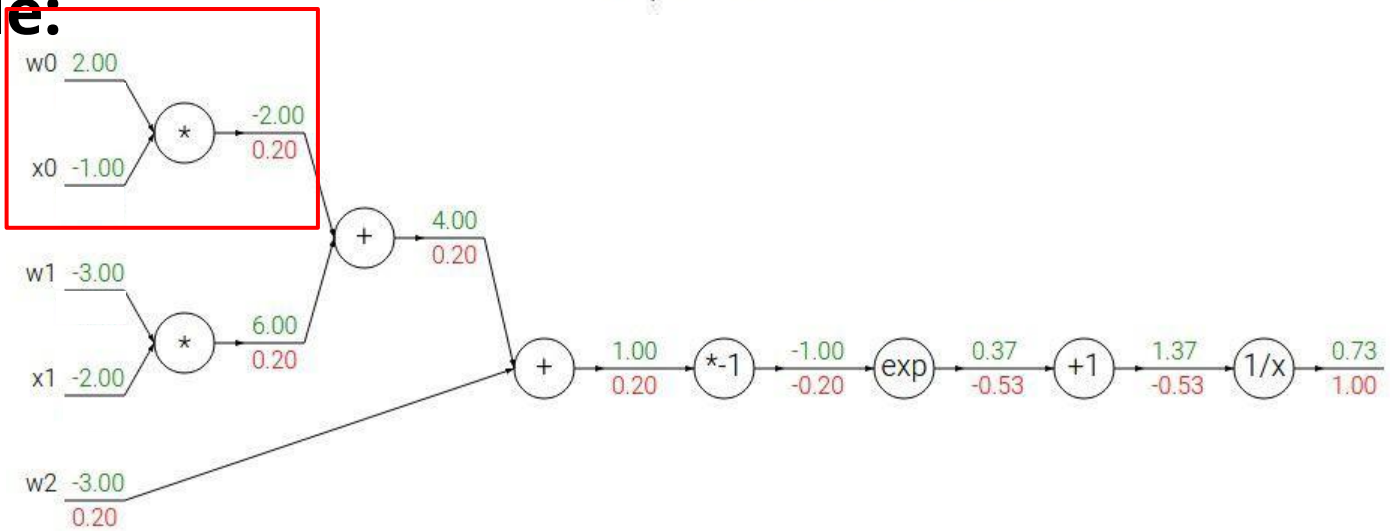
$f(x) = \frac{1}{x} \rightarrow \frac{df}{dx} = -1/x^2$

$f_c(x) = c + x \rightarrow \frac{df}{dx} = 1$

Credit to Stanford CS 231n

Another example:

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



$$f(x) = e^x \rightarrow \frac{df}{dx} = e^x$$

$$f_a(x) = ax \rightarrow \frac{df}{dx} = a$$

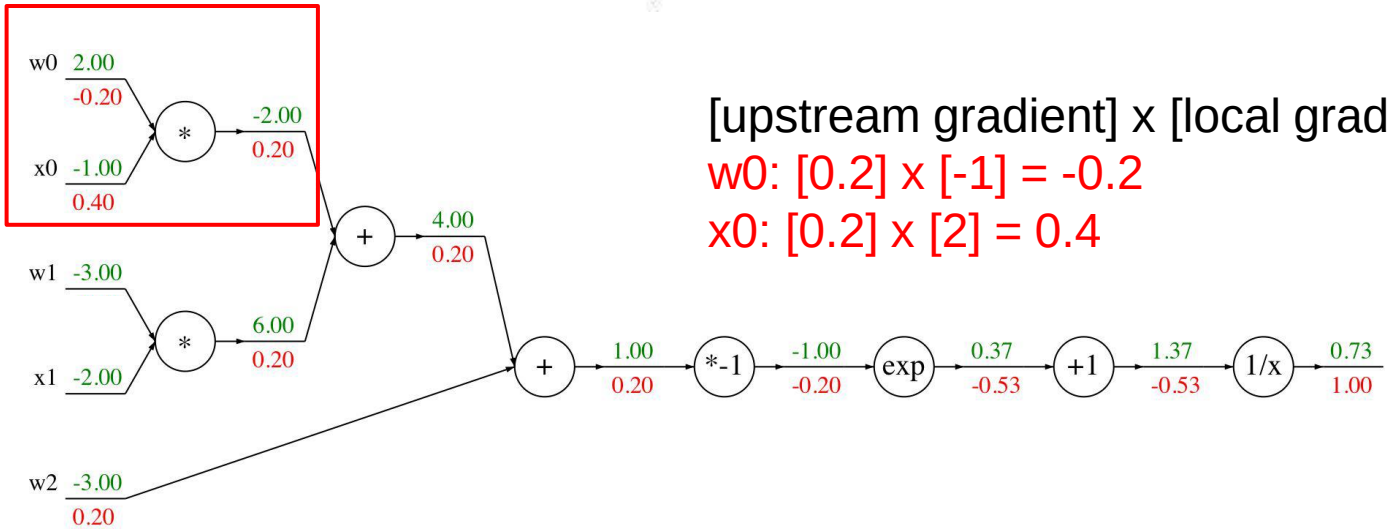
$$f(x) = \frac{1}{x} \rightarrow \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \rightarrow \frac{df}{dx} = 1$$

Credit to Stanford CS 231n

Another example:

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



[upstream gradient] x [local gradient]

w0: [0.2] x [-1] = -0.2

x0: [0.2] x [2] = 0.4

$$f(x) = e^x \rightarrow \frac{df}{dx} = e^x$$

$$f_a(x) = ax \rightarrow \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \rightarrow \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \rightarrow \frac{df}{dx} = 1$$

Credit to Stanford CS 231n

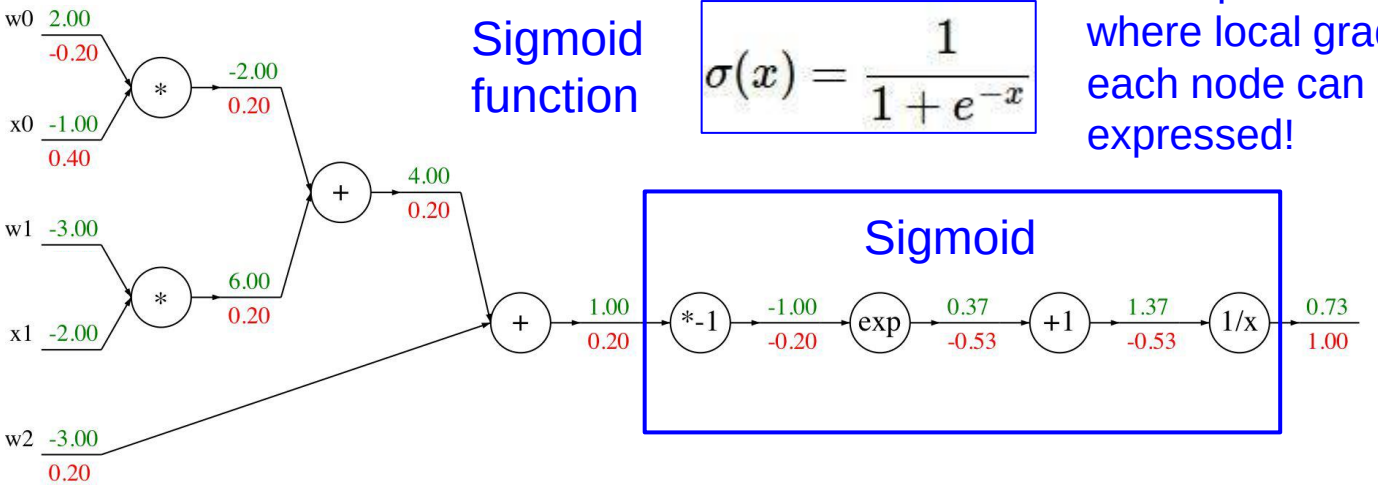
Another example:

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$

Computational graph representation may not be unique. Choose one where local gradients at each node can be easily expressed!

Sigmoid function

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

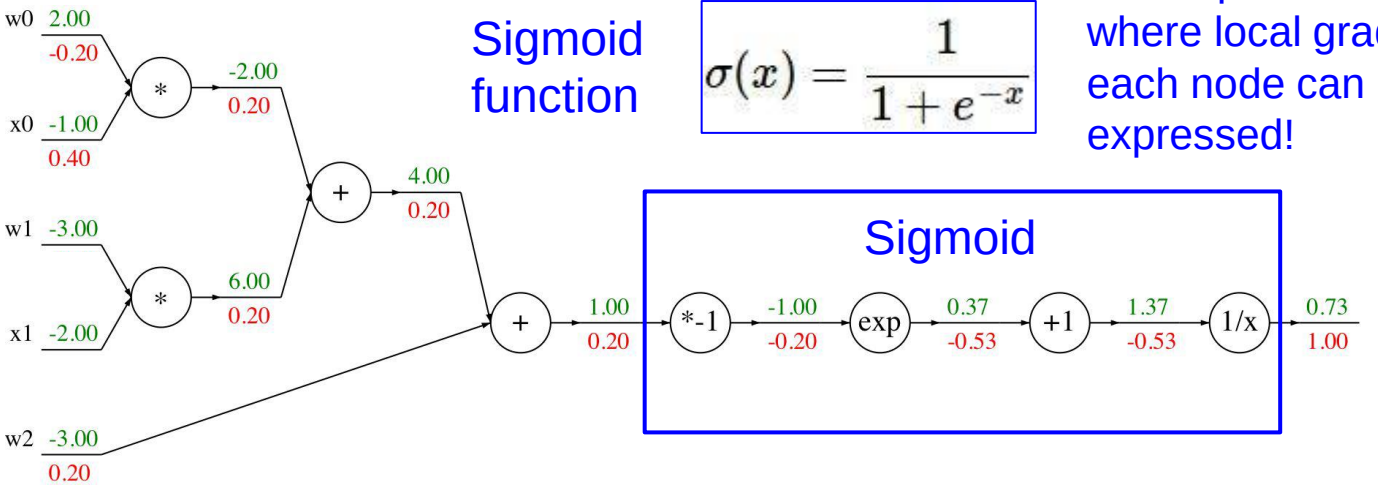


Credit to Stanford CS 231n

Another example:

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$

Computational graph representation may not be unique. Choose one where local gradients at each node can be easily expressed!



Sigmoid function

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

Sigmoid local gradient:

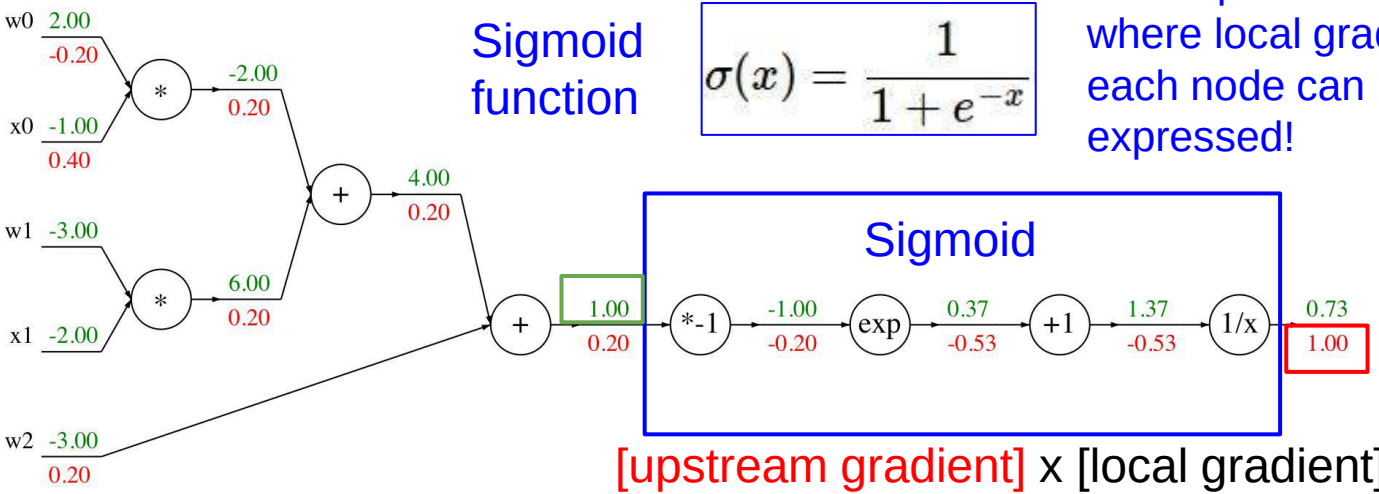
$$\frac{d\sigma(x)}{dx} = \frac{e^{-x}}{(1+e^{-x})^2} = \left(\frac{1+e^{-x}-1}{1+e^{-x}}\right)\left(\frac{1}{1+e^{-x}}\right) = (1-\sigma(x))\,\sigma(x)$$

Credit to Stanford CS 231n

Another example:

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$

Computational graph representation may not be unique. Choose one where local gradients at each node can be easily expressed!



Sigmoid function

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

[upstream gradient] x [local gradient]

[1.00] x [(1 - 1/(1+e^{-1})) (1/(1+e^{-1}))] = 0.2

Sigmoid local gradient:

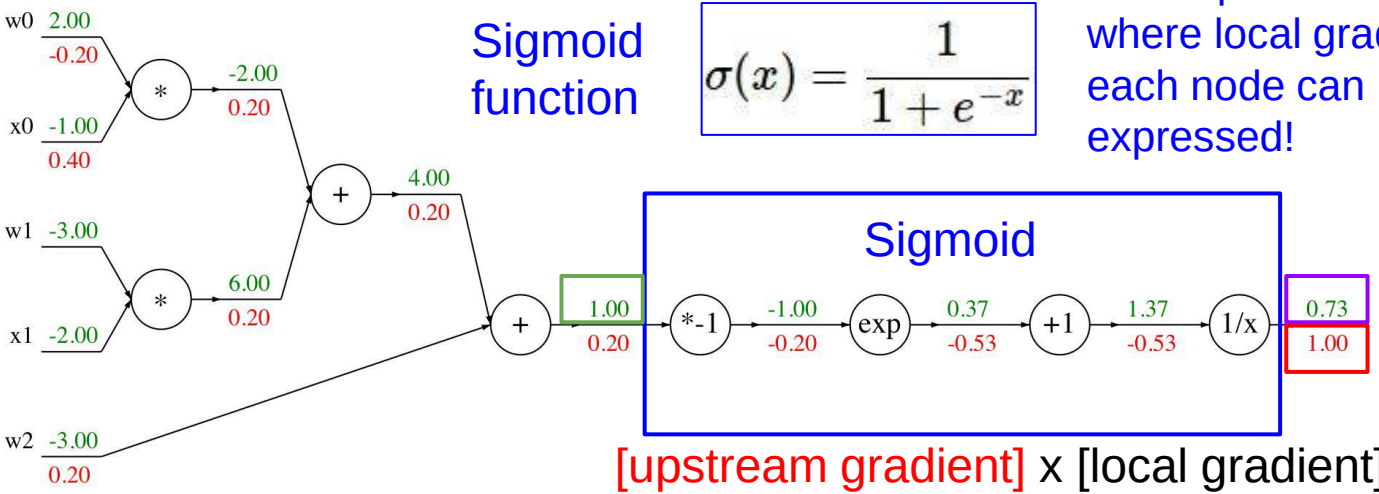
$$\frac{d\sigma(x)}{dx} = \frac{e^{-x}}{(1+e^{-x})^2} = \left(\frac{1+e^{-x}-1}{1+e^{-x}}\right)\left(\frac{1}{1+e^{-x}}\right) = (1-\sigma(x))\,\sigma(x)$$

Credit to Stanford CS 231n

Another example:

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$

Computational graph representation may not be unique. Choose one where local gradients at each node can be easily expressed!



Sigmoid function

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

[upstream gradient] x [local gradient]

[1.00] x [(1 - 0.73) (0.73)] = 0.2

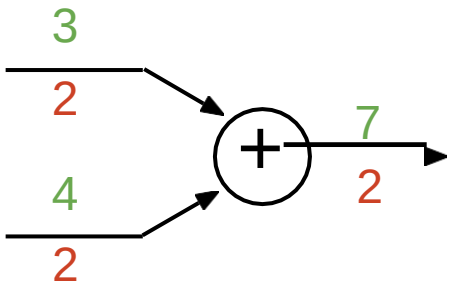
Sigmoid local gradient:

$$\frac{d\sigma(x)}{dx} = \frac{e^{-x}}{(1+e^{-x})^2} = \left(\frac{1+e^{-x}-1}{1+e^{-x}}\right)\left(\frac{1}{1+e^{-x}}\right) = (1-\sigma(x))\,\sigma(x)$$

Credit to Stanford CS 231n

Patterns in gradient flow

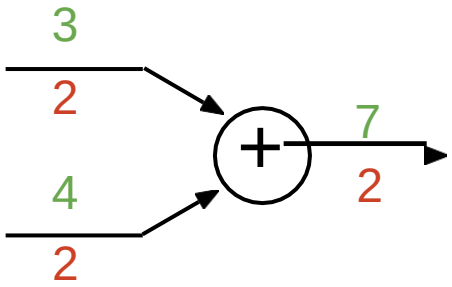
add gate: gradient distributor



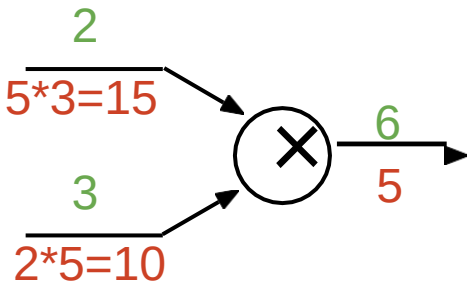
Credit to Stanford CS 231n

Patterns in gradient flow

add gate: gradient distributor



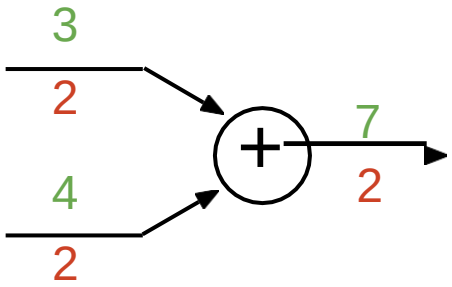
mul gate: “swap multiplier”



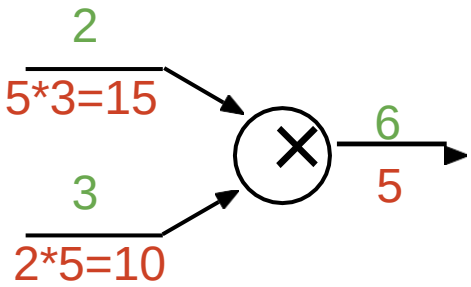
Credit to Stanford CS 231n

Patterns in gradient flow

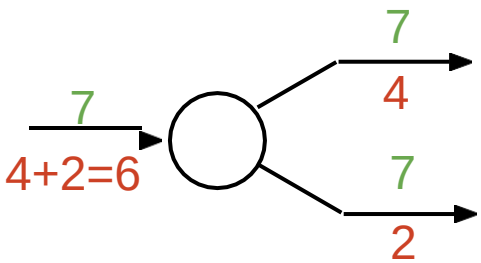
add gate: gradient distributor



mul gate: “swap multiplier”



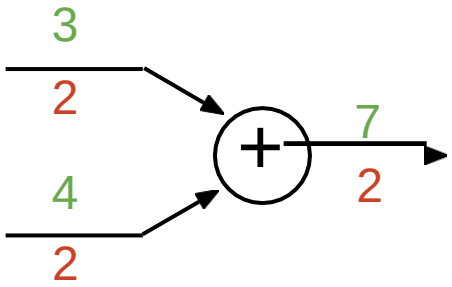
copy gate: gradient adder



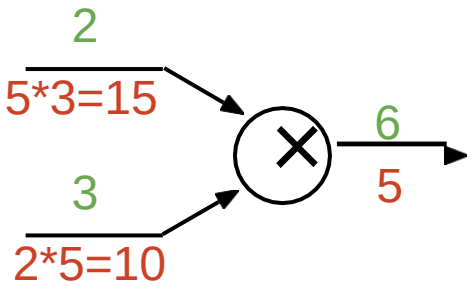
Credit to Stanford CS 231n

Patterns in gradient flow

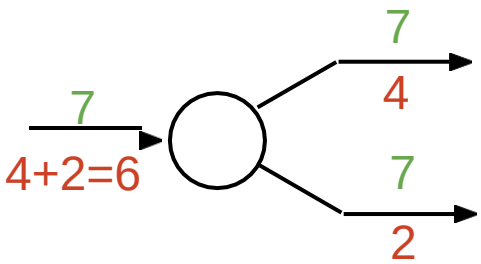
add gate: gradient distributor



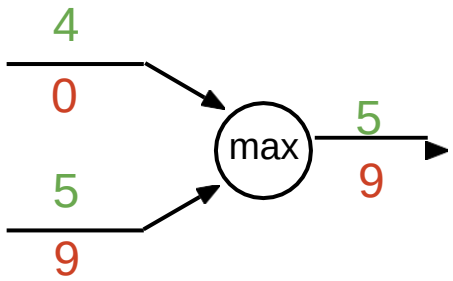
mul gate: "swap multiplier"



copy gate: gradient adder



max gate: gradient router

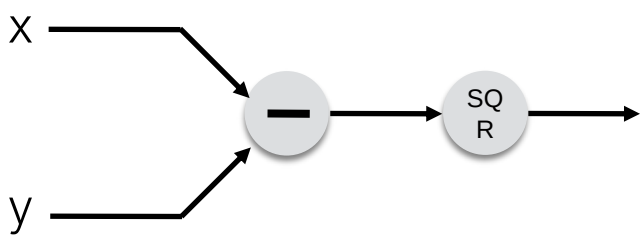


Credit to Stanford CS 231n

Computational Graphs

L2-loss

$$L_2(x, y) = (x - y)^2$$

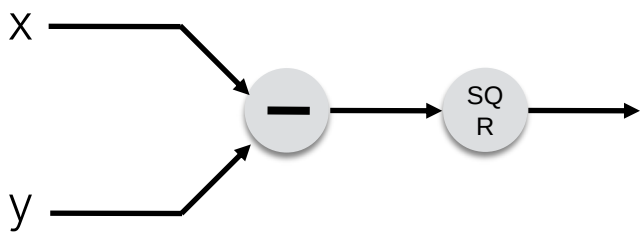


Credit to Stanford CS 231n

Computational Graphs

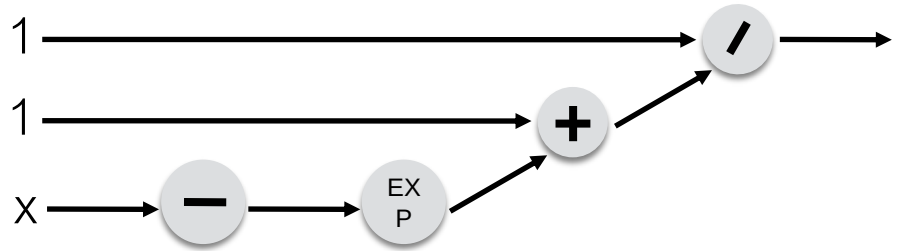
L2-loss

$$L_2(x, y) = (x - y)^2$$



Sigmoid

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

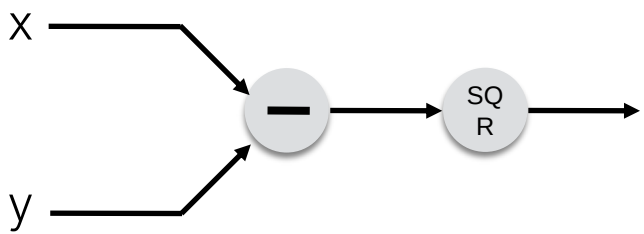


Credit to Stanford CS 231n

Computational Graphs

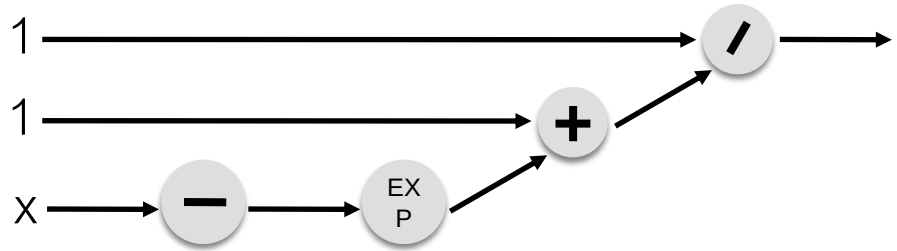
L2-loss

$$L_2(x, y) = (x - y)^2$$



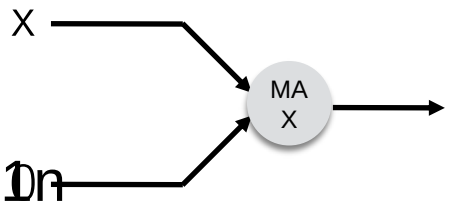
Sigmoid

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



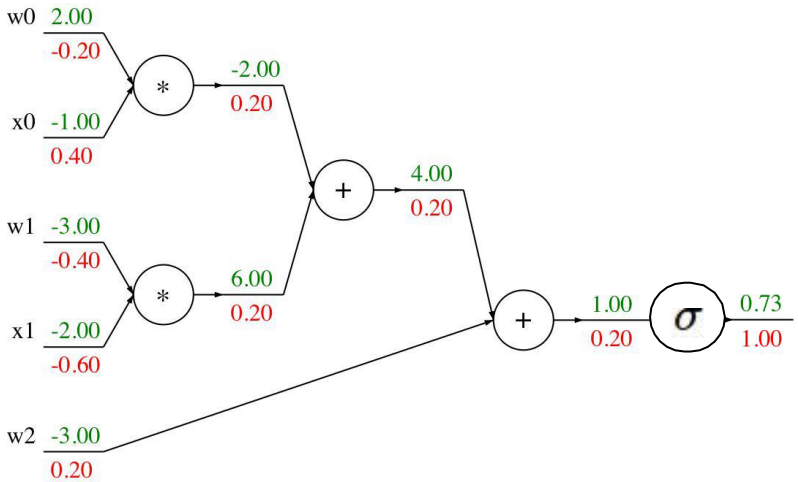
ReLU

$$\text{ReLU}(x) = \max(x, 0)$$



Credit to Stanford CS 231n

Backprop Implementation: "Flat" code



Forward pass:
Compute output

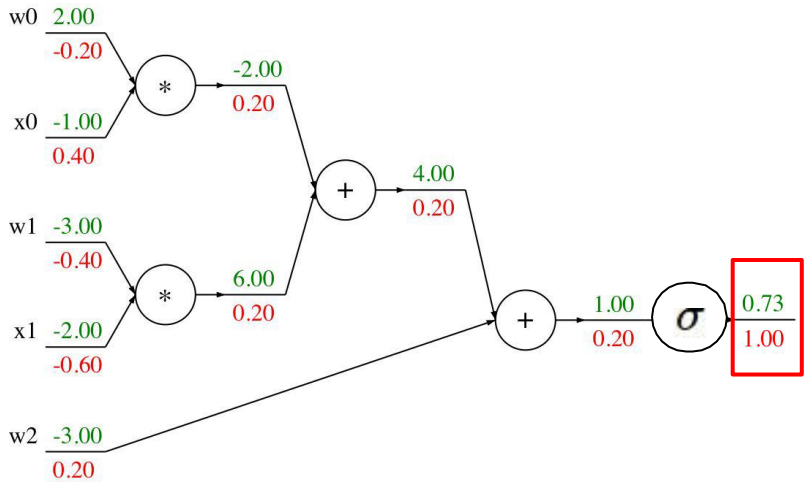
```
def f(w0, x0, w1, x1, w2):
  s0 = w0 * x0
  s1 = w1 * x1
  s2 = s0 + s1
  s3 = s2 + w2
  L = sigmoid(s3)
```

Backward pass:
Compute grads

```
grad_L = 1.0
grad_s3 = grad_L * (1 - L) * L
grad_w2 = grad_s3
grad_s2 = grad_s3
grad_s0 = grad_s2
grad_s1 = grad_s2
grad_w1 = grad_s1 * x1
grad_x1 = grad_s1 * w1
grad_w0 = grad_s0 * x0
grad_x0 = grad_s0 * w0
```

Credit to Stanford CS 231n

Backprop Implementation: "Flat" code



Forward pass: Compute output

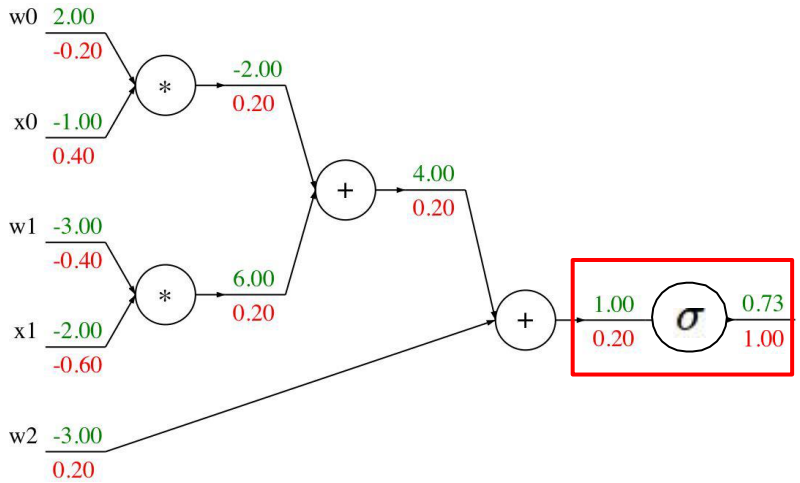
```
def f(w0, x0, w1, x1, w2):
    s0 = w0 * x0
    s1 = w1 * x1
    s2 = s0 + s1
    s3 = s2 + w2
    L = sigmoid(s3)
```

Base case

```
    grad_L = 1.0
    grad_s3 = grad_L * (1 - L) * L
    grad_w2 = grad_s3
    grad_s2 = grad_s3
    grad_s0 = grad_s2
    grad_s1 = grad_s2
    grad_w1 = grad_s1 * x1
    grad_x1 = grad_s1 * w1
    grad_w0 = grad_s0 * x0
    grad_x0 = grad_s0 * w0
```

Credit to Stanford CS 231n

Backprop Implementation: "Flat" code



Forward pass:
Compute output

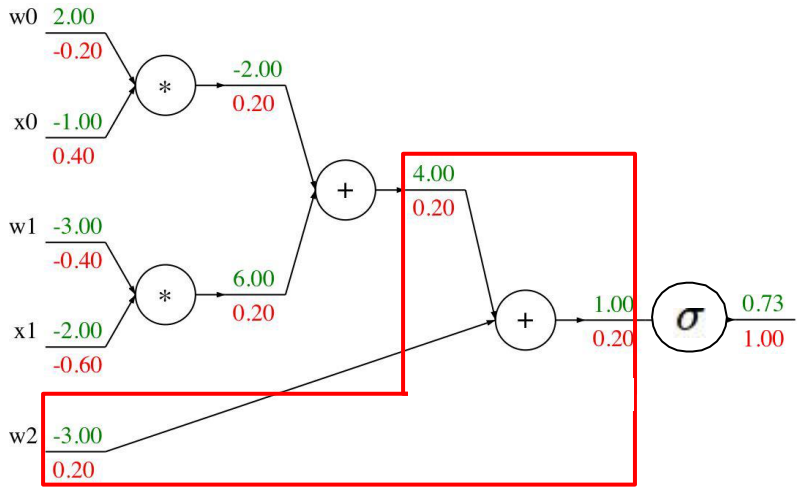
```
def f(w0, x0, w1, x1, w2):
  s0 = w0 * x0
  s1 = w1 * x1
  s2 = s0 + s1
  s3 = s2 + w2
  L = sigmoid(s3)
```

Sigmoid

```
grad_L = 1.0
grad_s3 = grad_L * (1 - L) * L
grad_w2 = grad_s3
grad_s2 = grad_s3
grad_s0 = grad_s2
grad_s1 = grad_s2
grad_w1 = grad_s1 * x1
grad_x1 = grad_s1 * w1
grad_w0 = grad_s0 * x0
grad_x0 = grad_s0 * w0
```

Credit to Stanford CS 231n

Backprop Implementation: "Flat" code



Forward pass:
Compute output

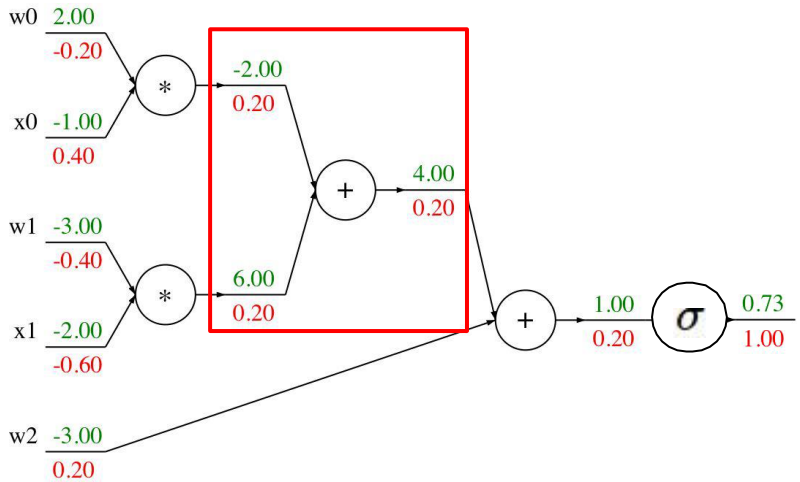
```
def f(w0, x0, w1, x1, w2):
  s0 = w0 * x0
  s1 = w1 * x1
  s2 = s0 + s1
  s3 = s2 + w2
  L = sigmoid(s3)
```

Add gate

```
grad_L = 1.0
grad_s3 = grad_L * (1 - L) * L
grad_w2 = grad_s3
grad_s2 = grad_s3
grad_s0 = grad_s2
grad_s1 = grad_s2
grad_w1 = grad_s1 * x1
grad_x1 = grad_s1 * w1
grad_w0 = grad_s0 * x0
grad_x0 = grad_s0 * w0
```

Credit to Stanford CS 231n

Backprop Implementation: "Flat" code



Forward pass:
Compute output

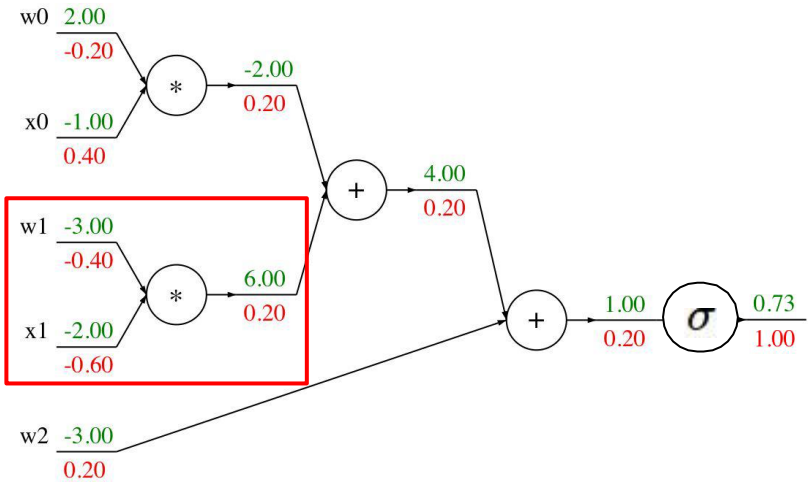
```
def f(w0, x0, w1, x1, w2):
  s0 = w0 * x0
  s1 = w1 * x1
  s2 = s0 + s1
  s3 = s2 + w2
  L = sigmoid(s3)
```

Add gate

```
  grad_L = 1.0
  grad_s3 = grad_L * (1 - L) * L
  grad_w2 = grad_s3
  grad_s2 = grad_s3
  grad_s0 = grad_s2
  grad_s1 = grad_s2
  grad_w1 = grad_s1 * x1
  grad_x1 = grad_s1 * w1
  grad_w0 = grad_s0 * x0
  grad_x0 = grad_s0 * w0
```

Credit to Stanford CS 231n

Backprop Implementation: "Flat" code



Forward pass:
Compute output

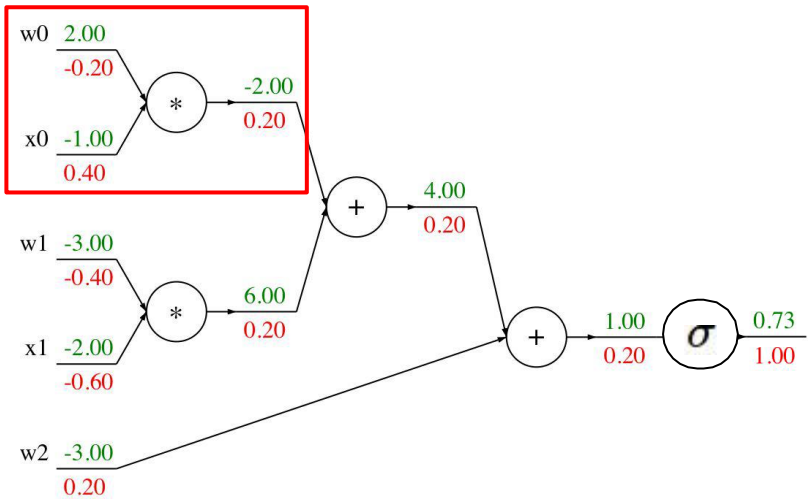
```
def f(w0, x0, w1, x1, w2):
  s0 = w0 * x0
  s1 = w1 * x1
  s2 = s0 + s1
  s3 = s2 + w2
  L = sigmoid(s3)
```

```
grad_L = 1.0
grad_s3 = grad_L * (1 - L) * L
grad_w2 = grad_s3
grad_s2 = grad_s3
grad_s0 = grad_s2
grad_s1 = grad_s2
grad_w1 = grad_s1 * x1
grad_x1 = grad_s1 * w1
grad_w0 = grad_s0 * x0
grad_x0 = grad_s0 * w0
```

Multiply gate

Credit to Stanford CS 231n

Backprop Implementation: "Flat" code



Forward pass:
Compute output

```
def f(w0, x0, w1, x1, w2):
  s0 = w0 * x0
  s1 = w1 * x1
  s2 = s0 + s1
  s3 = s2 + w2
  L = sigmoid(s3)
```

```
  grad_L = 1.0
  grad_s3 = grad_L * (1 - L) * L
  grad_w2 = grad_s3
  grad_s2 = grad_s3
  grad_s0 = grad_s2
  grad_s1 = grad_s2
  grad_w1 = grad_s1 * x1
  grad_x1 = grad_s1 * w1
  grad_w0 = grad_s0 * x0
  grad_x0 = grad_s0 * w0
```

Multiply gate

Credit to Stanford CS 231n

Vector derivatives

Scalar to Scalar

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

Regular derivative:

$$\frac{\partial y}{\partial x} \in \mathbb{R}$$

If x changes by a small amount, how much will y change?

Credit to Stanford CS 231n

Vector derivatives

Scalar to Scalar

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

Regular derivative:

$$\frac{\partial y}{\partial x} \in \mathbb{R}$$

If x changes by a small amount, how much will y change?

Vector to Scalar

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

Derivative is **Gradient**:

$$\frac{\partial y}{\partial x} \in \mathbb{R}^N \quad \left(\frac{\partial y}{\partial x}\right)_n = \frac{\partial y}{\partial x_n}$$

For each element of x, if it changes by a small amount then how much will y change?

Credit to Stanford CS 231n

Vector derivatives

Scalar to Scalar

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

Regular derivative:

$$\frac{\partial y}{\partial x} \in \mathbb{R}$$

If x changes by a small amount, how much will y change?

Vector to Scalar

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

Derivative is **Gradient**:

$$\frac{\partial y}{\partial x} \in \mathbb{R}^N \quad \left(\frac{\partial y}{\partial x}\right)_n = \frac{\partial y}{\partial x_n}$$

For each element of x, if it changes by a small amount then how much will y change?

Vector to Vector

$x \in \mathbb{R}^N, y \in \mathbb{R}^M$

Derivative is **Jacobian**:

$$\frac{\partial y}{\partial x} \in \mathbb{R}^{N \times M} \quad \left(\frac{\partial y}{\partial x}\right)_{n,m} = \frac{\partial y_m}{\partial x_n}$$

For each element of x, if it changes by a small amount then how much will each element of y change?

Credit to Stanford CS 231n

Example 5.9 (Gradient of a Vector-Valued Function)
We are given

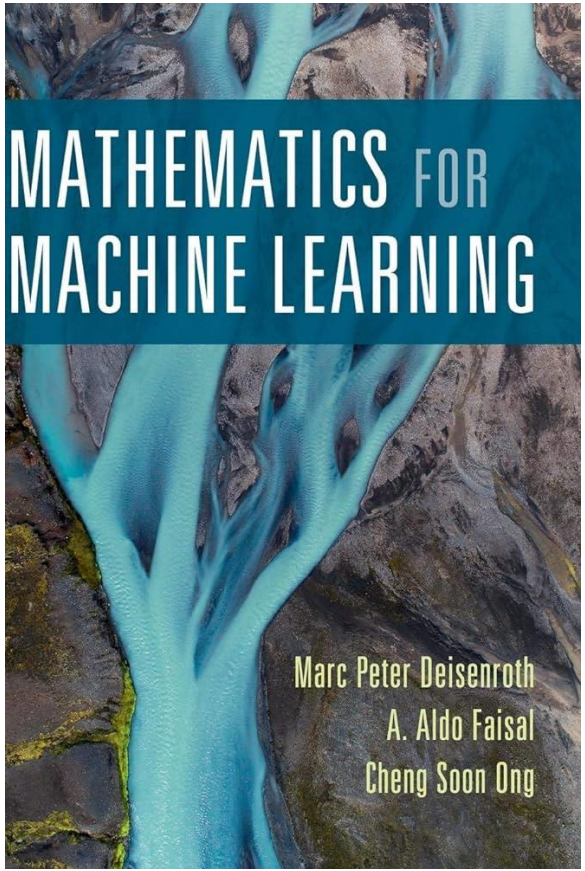
$$\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{A}\boldsymbol{x}\,, \qquad \boldsymbol{f}(\boldsymbol{x}) \in \mathbb{R}^M, \quad \boldsymbol{A} \in \mathbb{R}^{M\times N}, \quad \boldsymbol{x} \in \mathbb{R}^N\,.$$

To compute the gradient $\mathrm{d}\boldsymbol{f}/\mathrm{d}\boldsymbol{x}$ we first determine the dimension of $\mathrm{d}\boldsymbol{f}/\mathrm{d}\boldsymbol{x}$: Since $\boldsymbol{f} : \mathbb{R}^N \to \mathbb{R}^M$, it follows that $\mathrm{d}\boldsymbol{f}/\mathrm{d}\boldsymbol{x} \in \mathbb{R}^{M\times N}$. Second, to compute the gradient we determine the partial derivatives of f with respect to every x_j:

$$f_i(\boldsymbol{x}) = \sum_{j=1}^{N} A_{ij}x_j \implies \frac{\partial f_i}{\partial x_j} = A_{ij} \tag{5.67}$$

We collect the partial derivatives in the Jacobian and obtain the gradient

$$\frac{\mathrm{d}\boldsymbol{f}}{\mathrm{d}\boldsymbol{x}} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_N} \\ \vdots & & \vdots \\ \frac{\partial f_M}{\partial x_1} & \cdots & \frac{\partial f_M}{\partial x_N} \end{bmatrix} = \begin{bmatrix} A_{11} & \cdots & A_{1N} \\ \vdots & & \vdots \\ A_{M1} & \cdots & A_{MN} \end{bmatrix} = \boldsymbol{A} \in \mathbb{R}^{M\times N}\,. \tag{5.68}$$



Example 5.10 (Chain Rule)
Consider the function $h : \mathbb{R} \to \mathbb{R}$, $h(t) = (f \circ g)(t)$ with

$$f : \mathbb{R}^2 \to \mathbb{R} \tag{5.69}$$

$$g : \mathbb{R} \to \mathbb{R}^2 \tag{5.70}$$

$$f(\boldsymbol{x}) = \exp(x_1 x_2^2)\,, \tag{5.71}$$

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = g(t) = \begin{bmatrix} t\cos t \\ t \sin t \end{bmatrix} \tag{5.72}$$

and compute the gradient of h with respect to t. Since $f : \mathbb{R}^2 \to \mathbb{R}$ and $g : \mathbb{R} \to \mathbb{R}^2$ we note that

$$\frac{\partial f}{\partial \boldsymbol{x}} \in \mathbb{R}^{1\times 2}\,, \quad \frac{\partial g}{\partial t} \in \mathbb{R}^{2\times 1}\,. \tag{5.73}$$

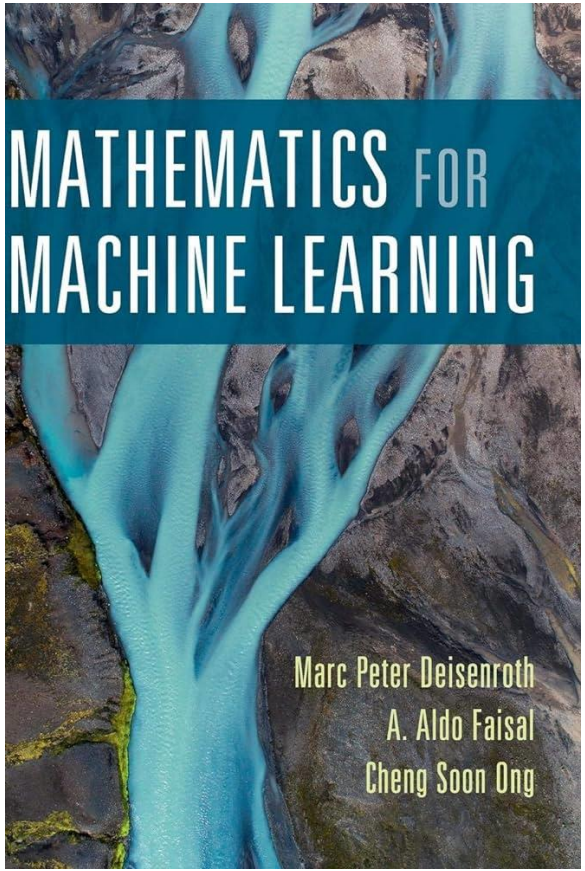
The desired gradient is computed by applying the chain rule:

$$\frac{\mathrm{d}h}{\mathrm{d}t} = \frac{\partial f}{\partial \boldsymbol{x}} \frac{\partial \boldsymbol{x}}{\partial t} = \begin{bmatrix} \frac{\partial f}{\partial x_1} & \frac{\partial f}{\partial x_2} \end{bmatrix} \begin{bmatrix} \frac{\partial x_1}{\partial t} \\ \frac{\partial x_2}{\partial t} \end{bmatrix} \tag{5.74a}$$

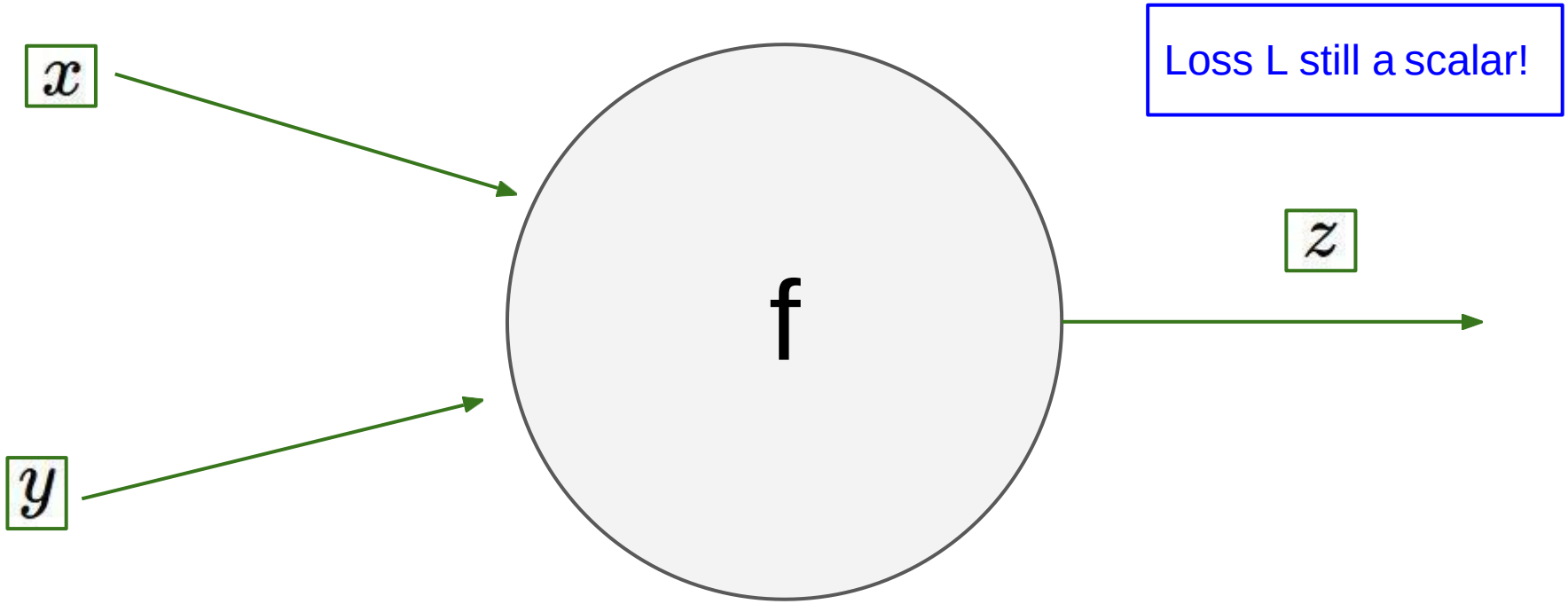
$$= \begin{bmatrix} \exp(x_1 x_2^2) x_2^2 & 2\exp(x_1 x_2^2) x_1 x_2 \end{bmatrix} \begin{bmatrix} \cos t - t \sin t \\ \sin t + t \cos t \end{bmatrix} \tag{5.74b}$$

$$= \exp(x_1 x_2^2)\big(x_2^2(\cos t - t \sin t) + 2x_1 x_2(\sin t + t \cos t)\big)\,, \tag{5.74c}$$

where $x_1 = t\cos t$ and $x_2 = t \sin t$; see (5.72).

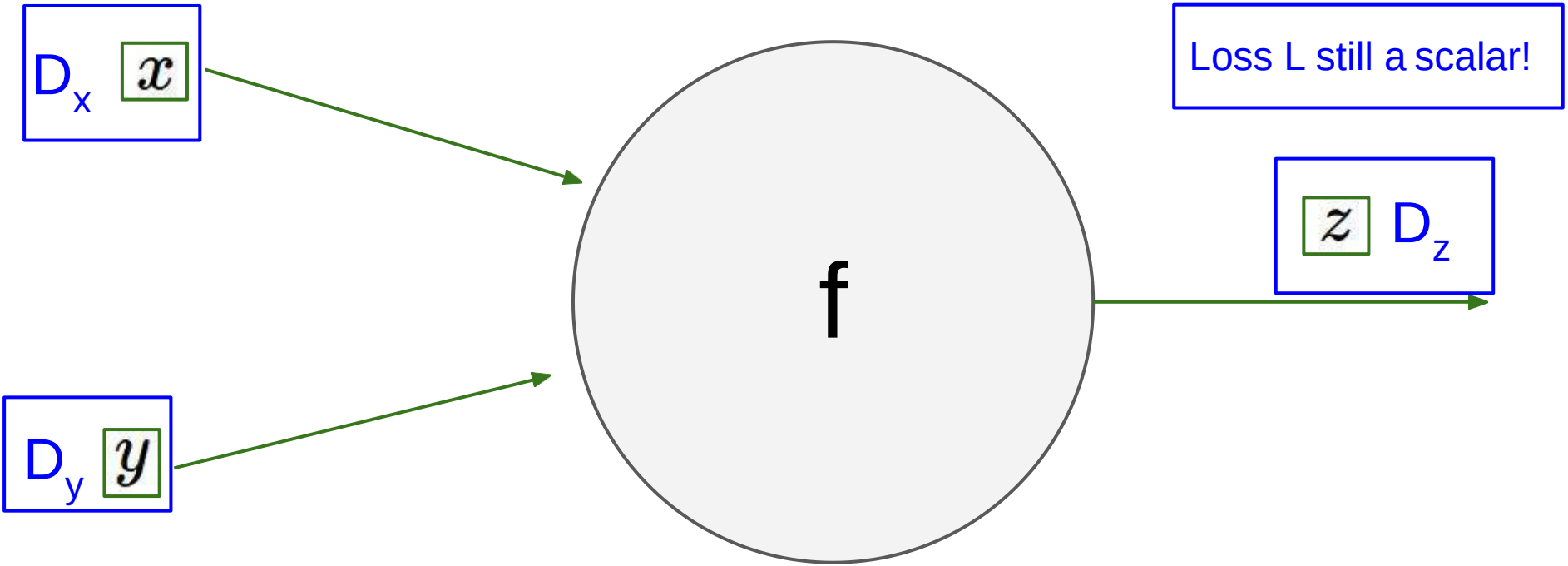


Backprop with Vectors



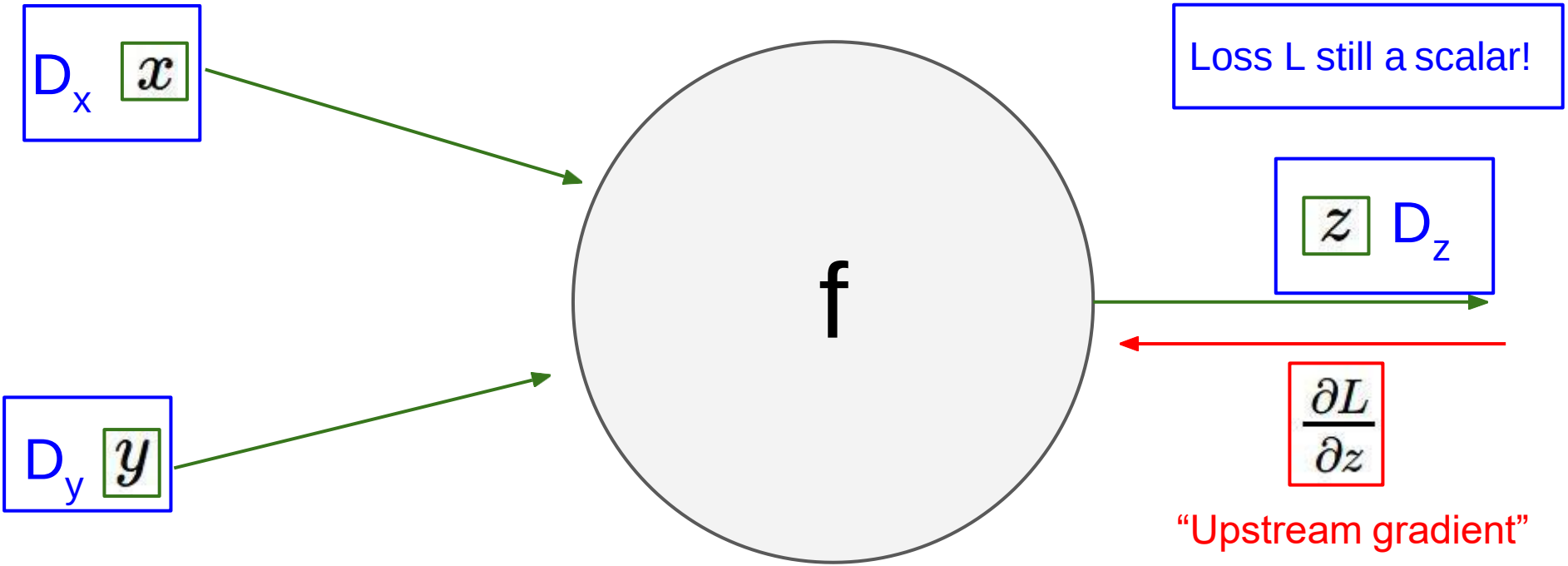
Credit to Stanford CS 231n

Backprop with Vectors



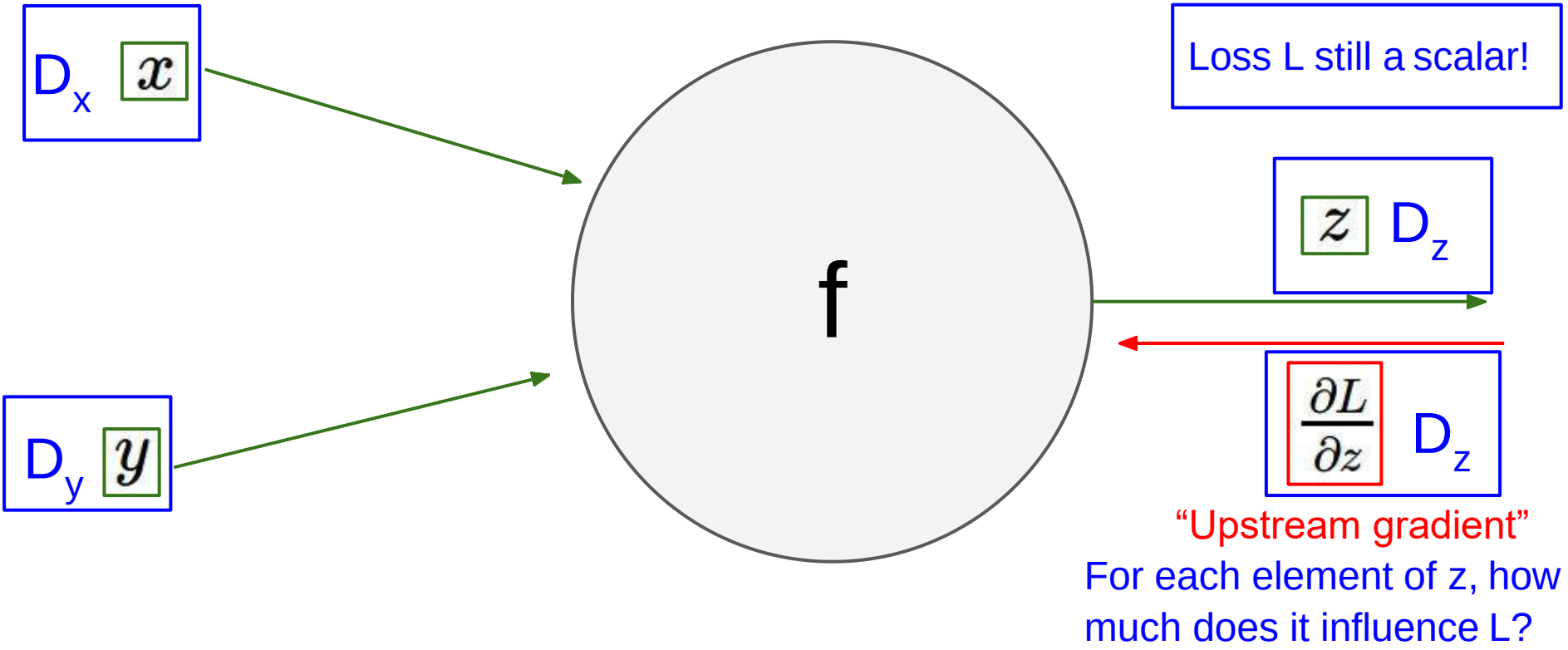
Credit to Stanford CS 231n

Backprop with Vectors



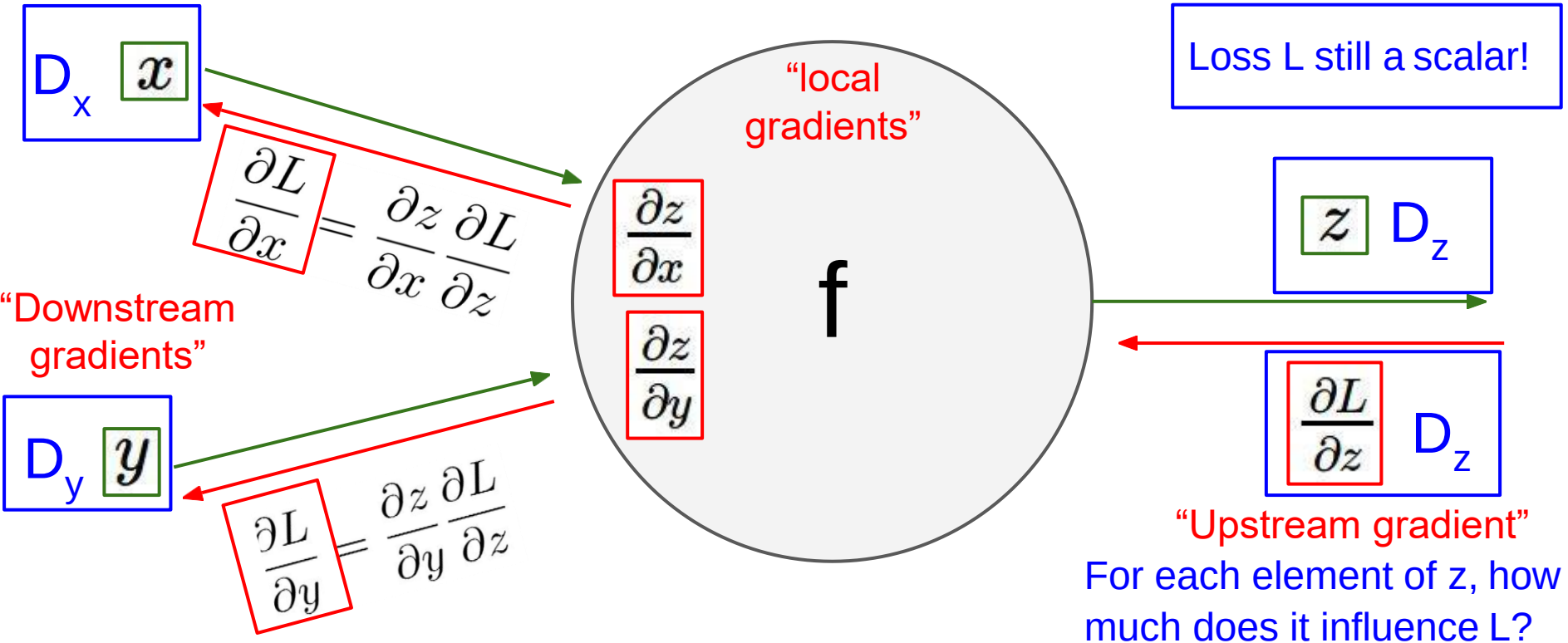
Credit to Stanford CS 231n

Backprop with Vectors



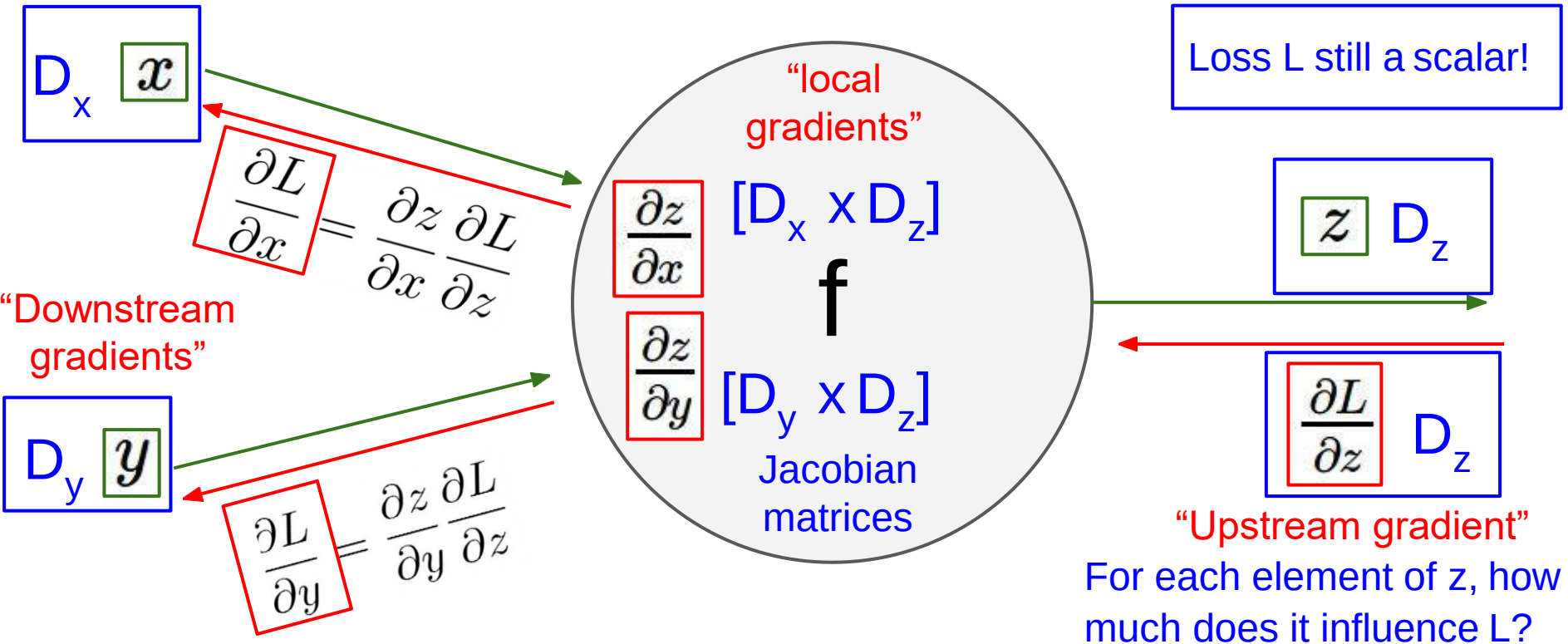
Credit to Stanford CS 231n

Backprop with Vectors



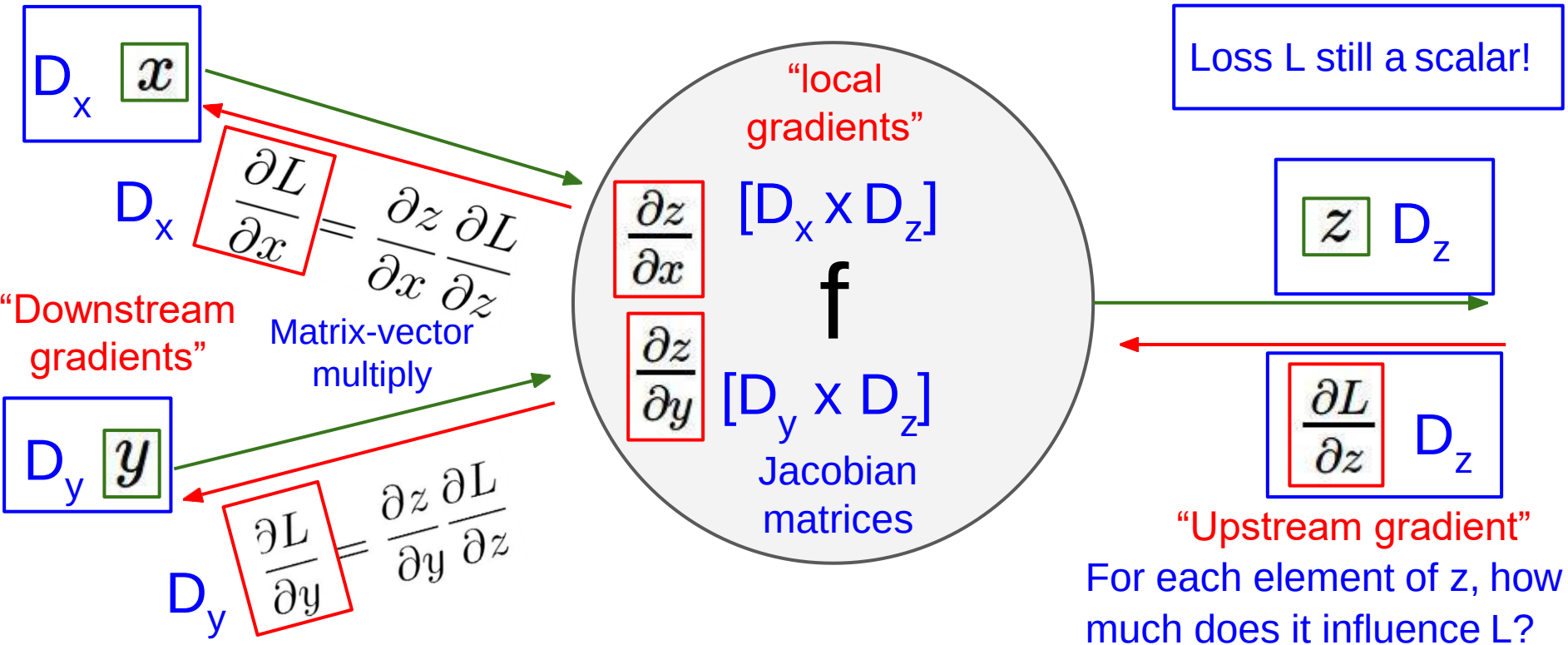
Credit to Stanford CS 231n

Backprop with Vectors



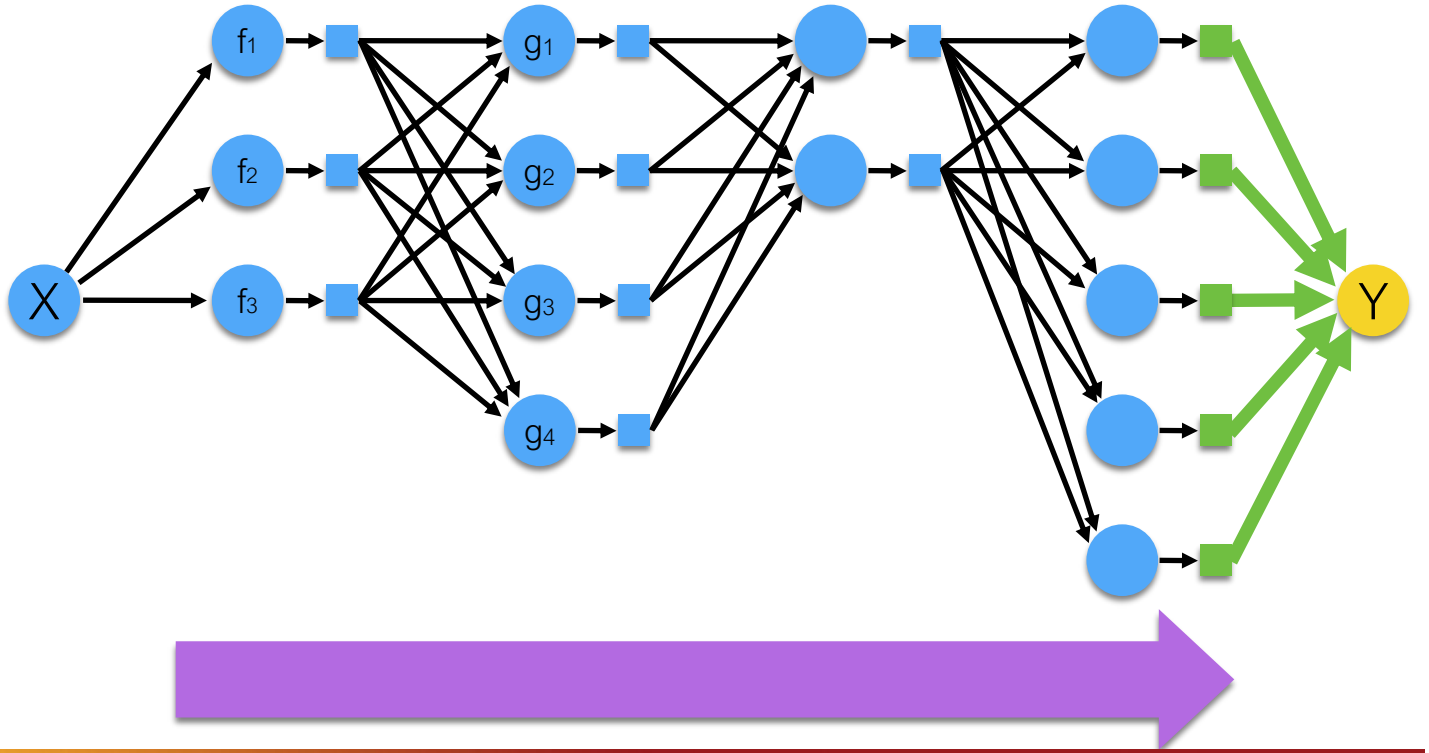
Credit to Stanford CS 231n

Backprop with Vectors

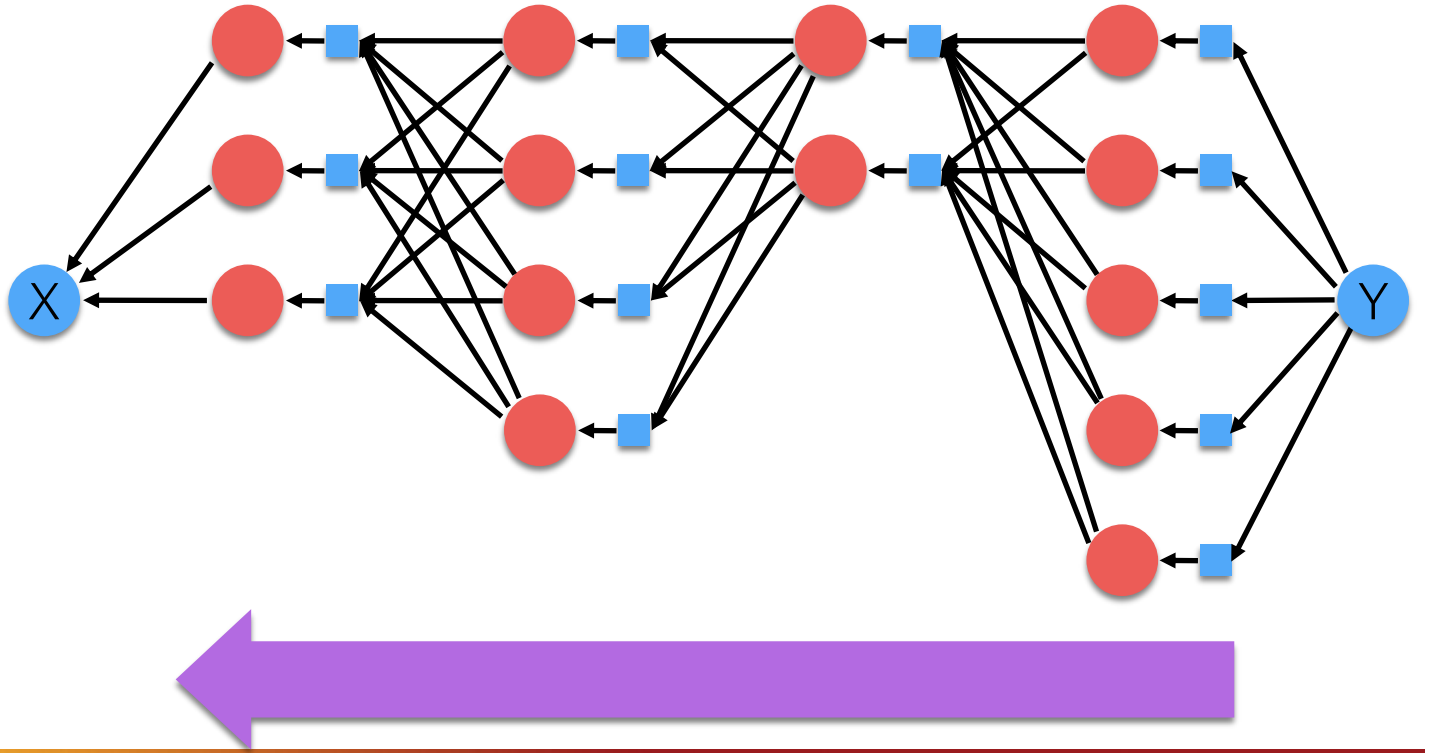


Credit to Stanford CS 231n

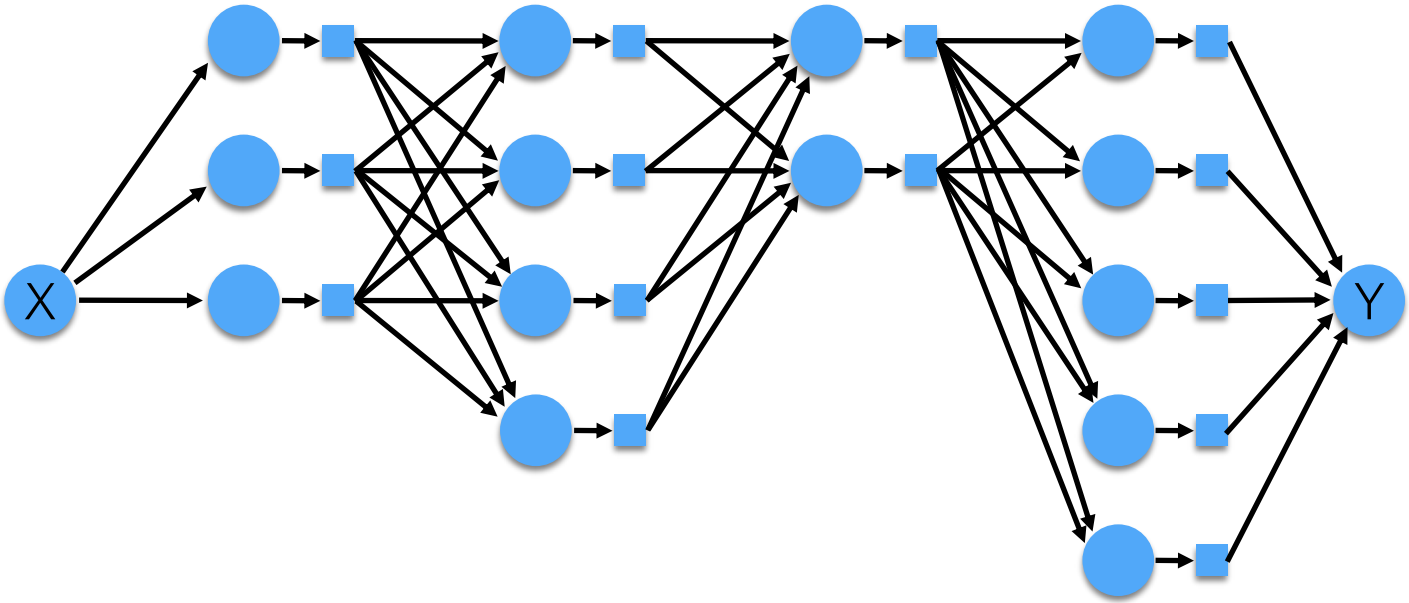
Forward Propagation



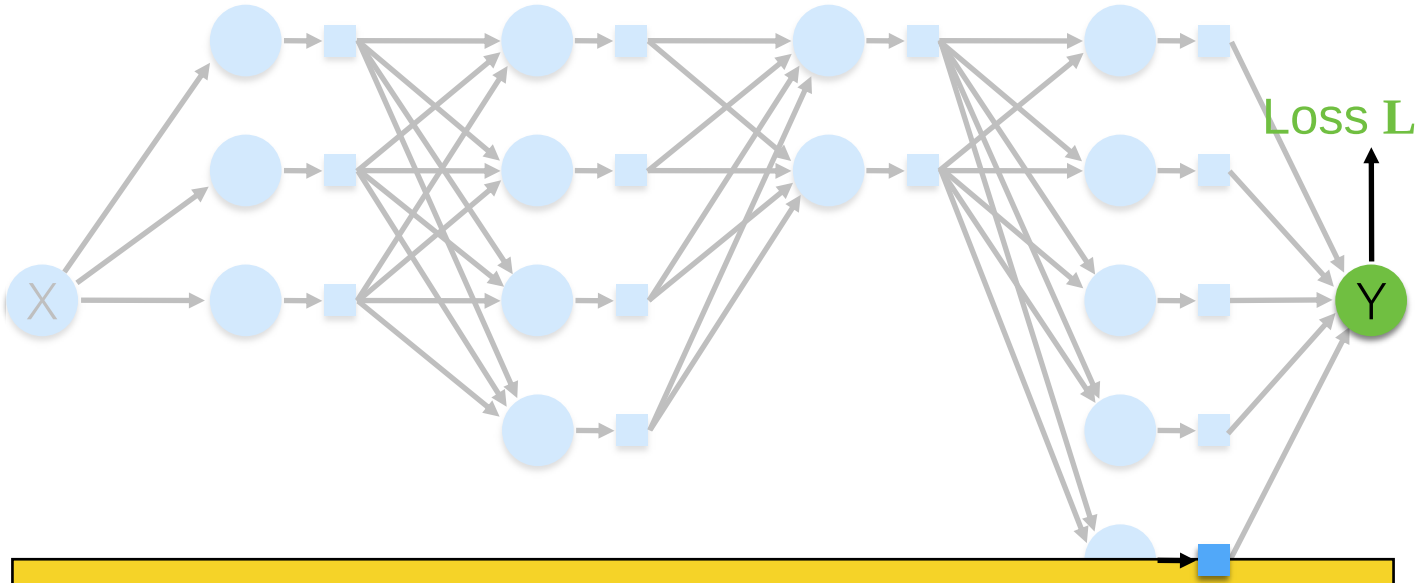
Back Propagation



Back Propagation



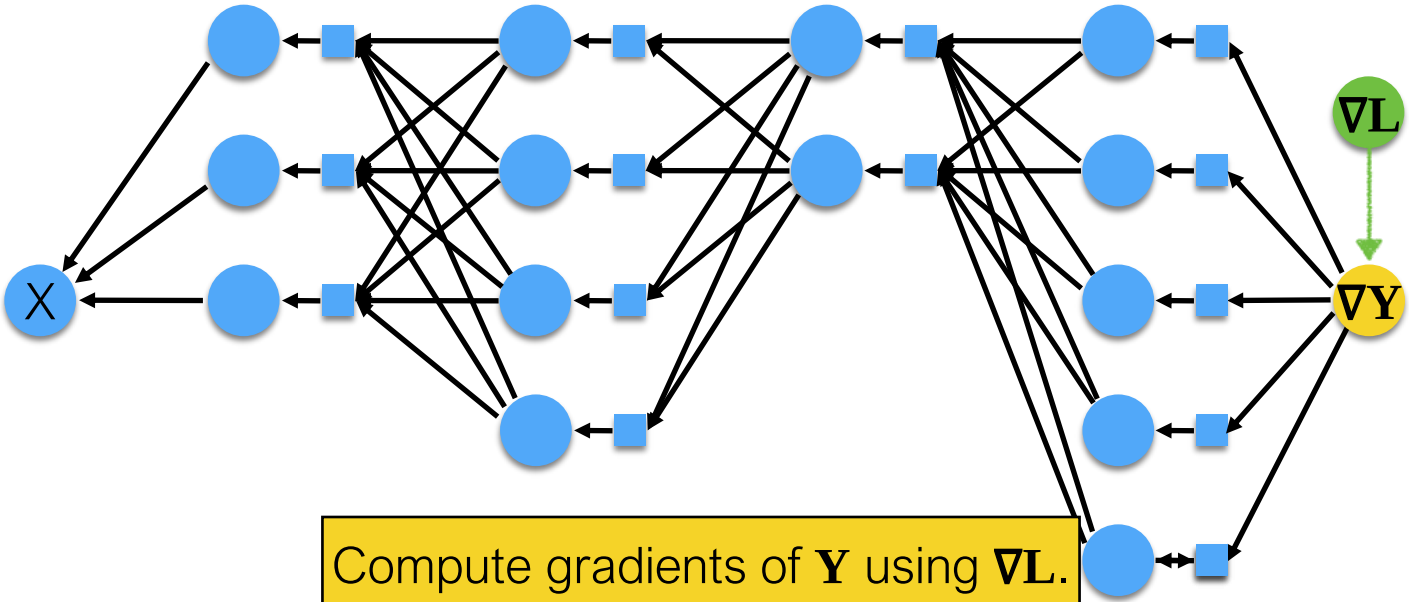
Back Propagation



Compute a loss using an estimated Y and the ground truth Y*.

e.g., $L(W) = \| f(X; W) - Y^* \|_2$

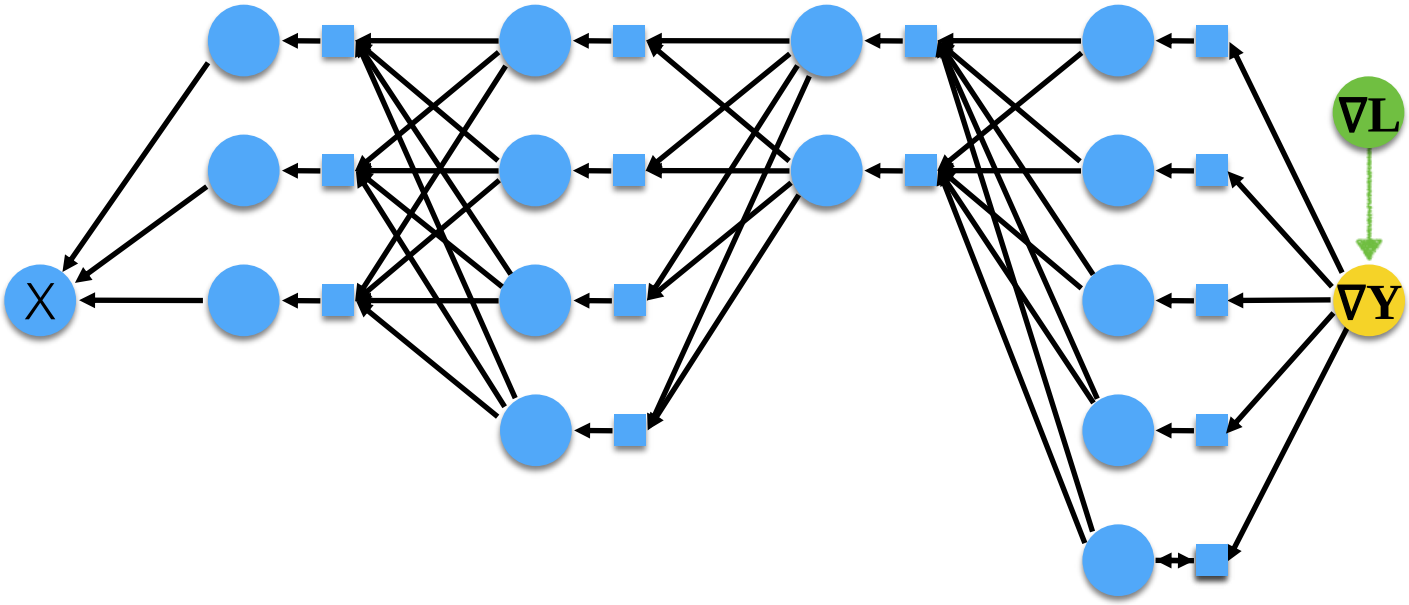
Back Propagation



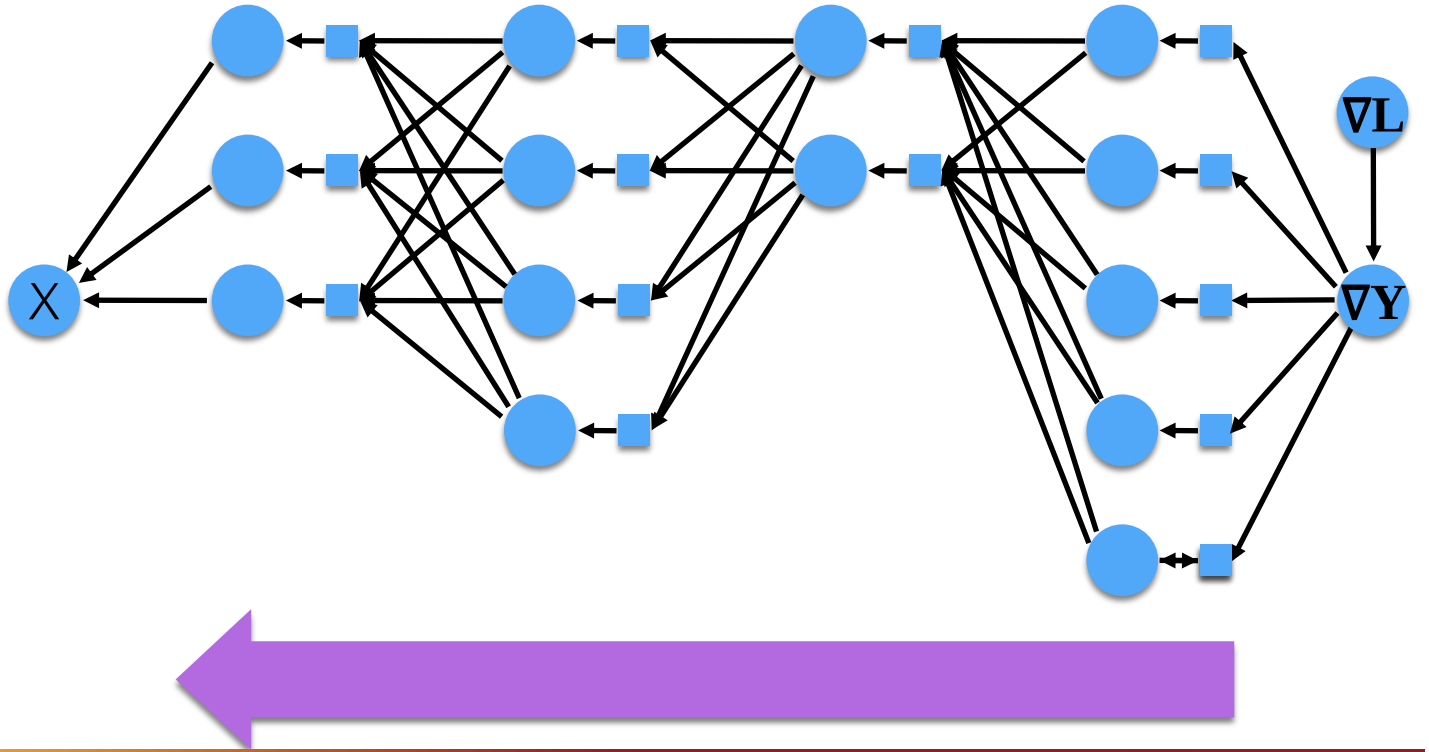
Compute gradients of $\mathbf{Y}$ using $\nabla\mathbf{L}$.

$$\frac{\partial L}{\partial X} = \frac{\partial L}{\partial Y} \cdot \frac{\partial Y}{\partial X}$$

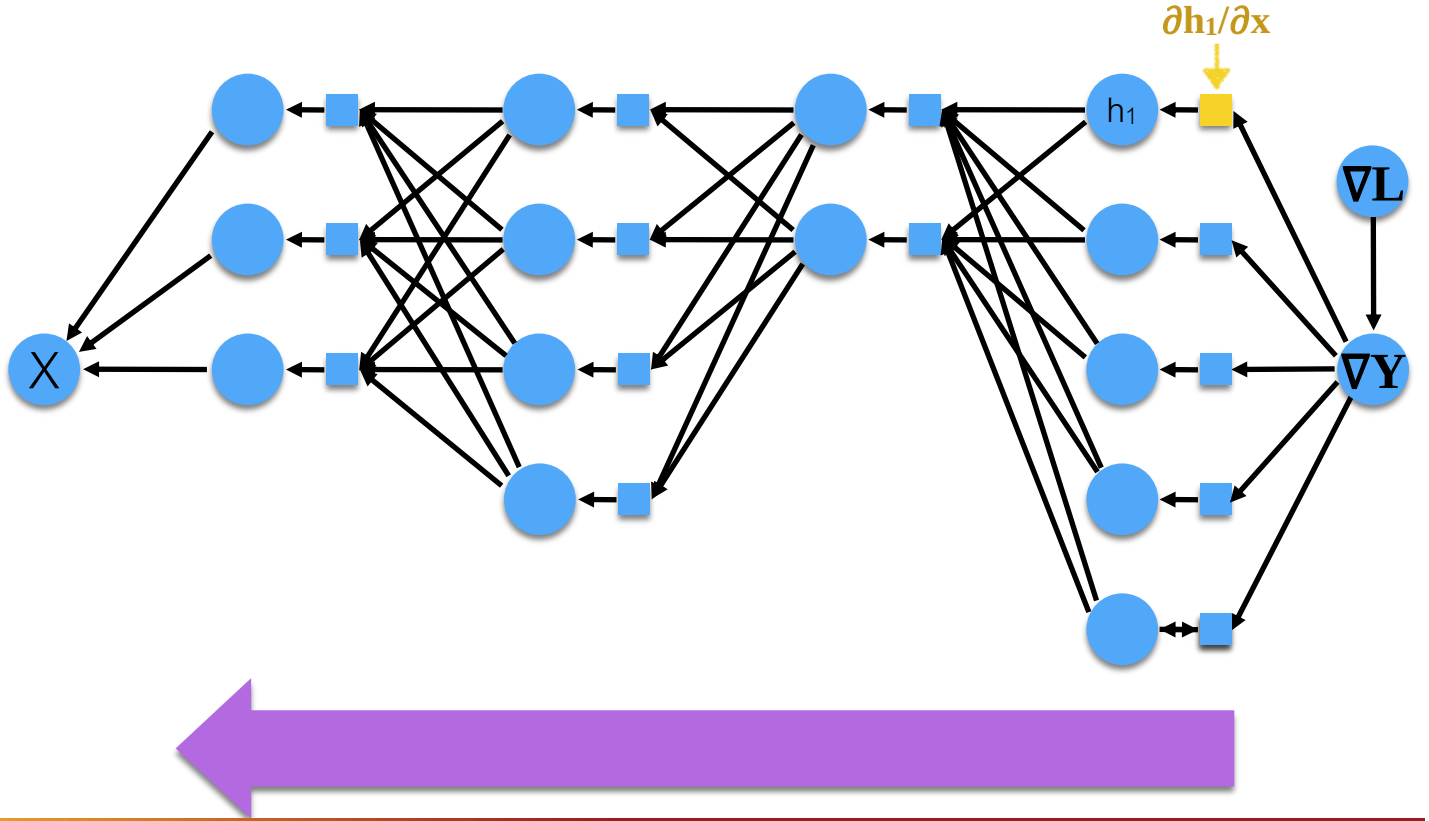
Back Propagation



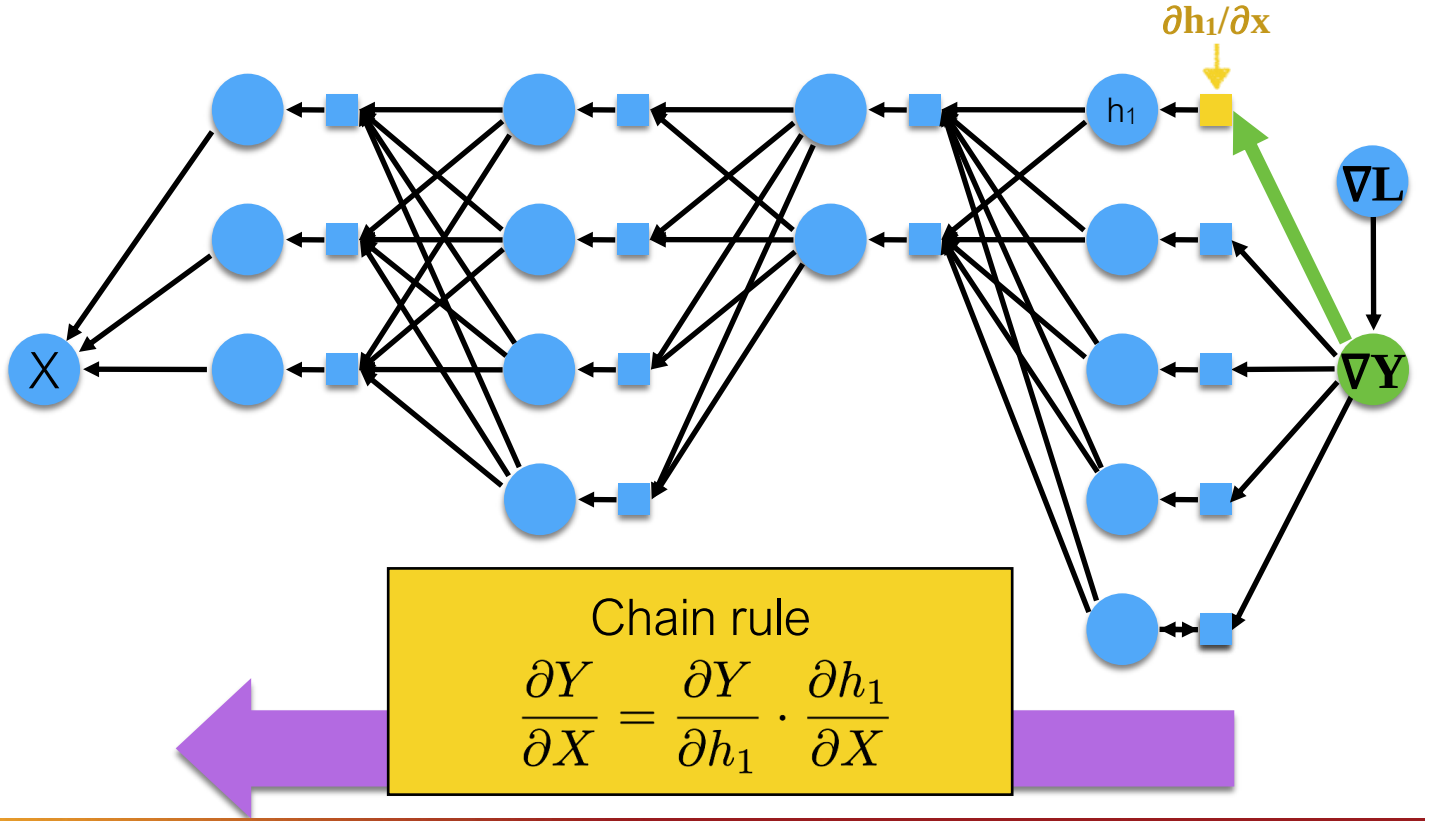
Back Propagation



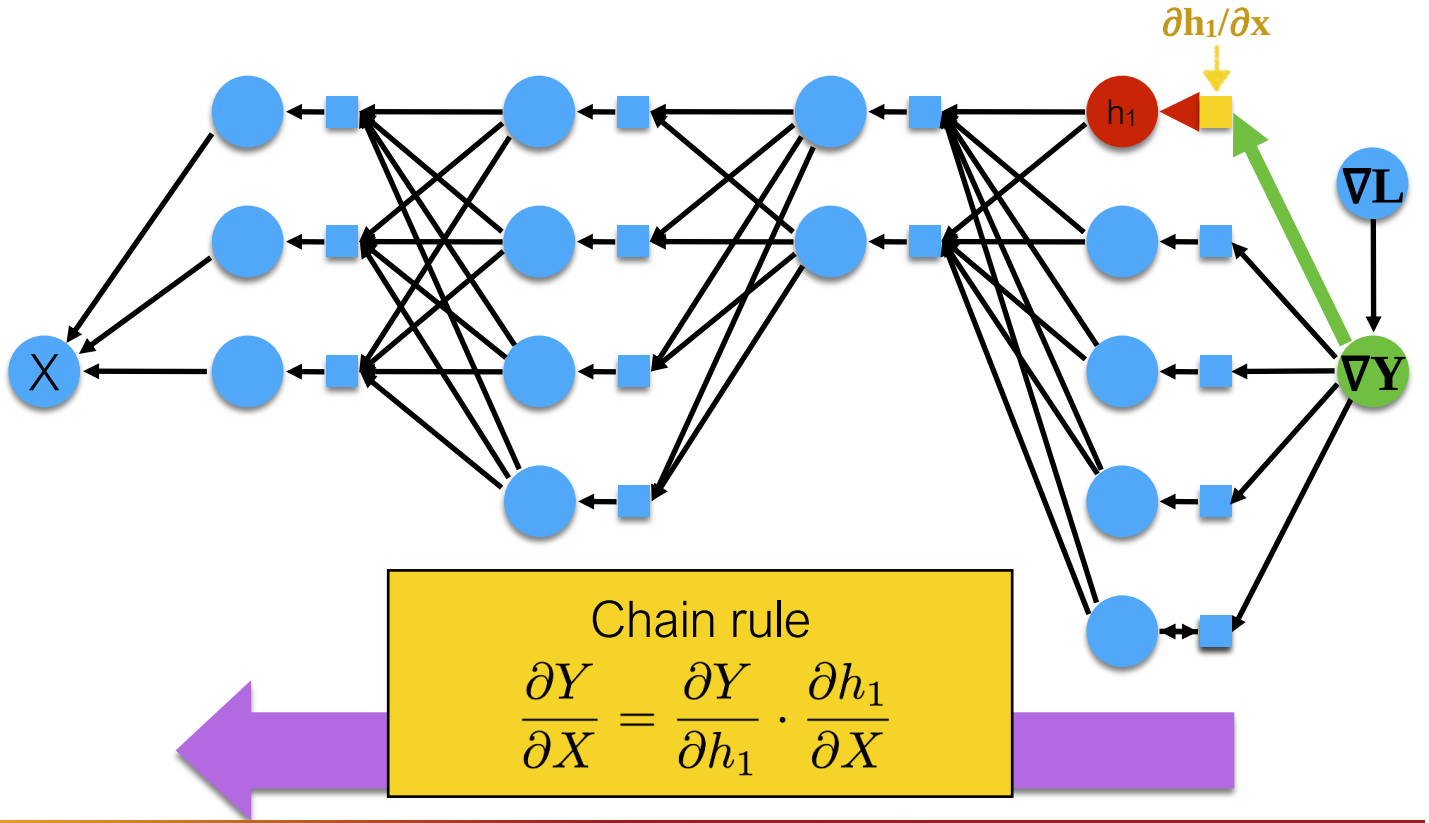
Back Propagation



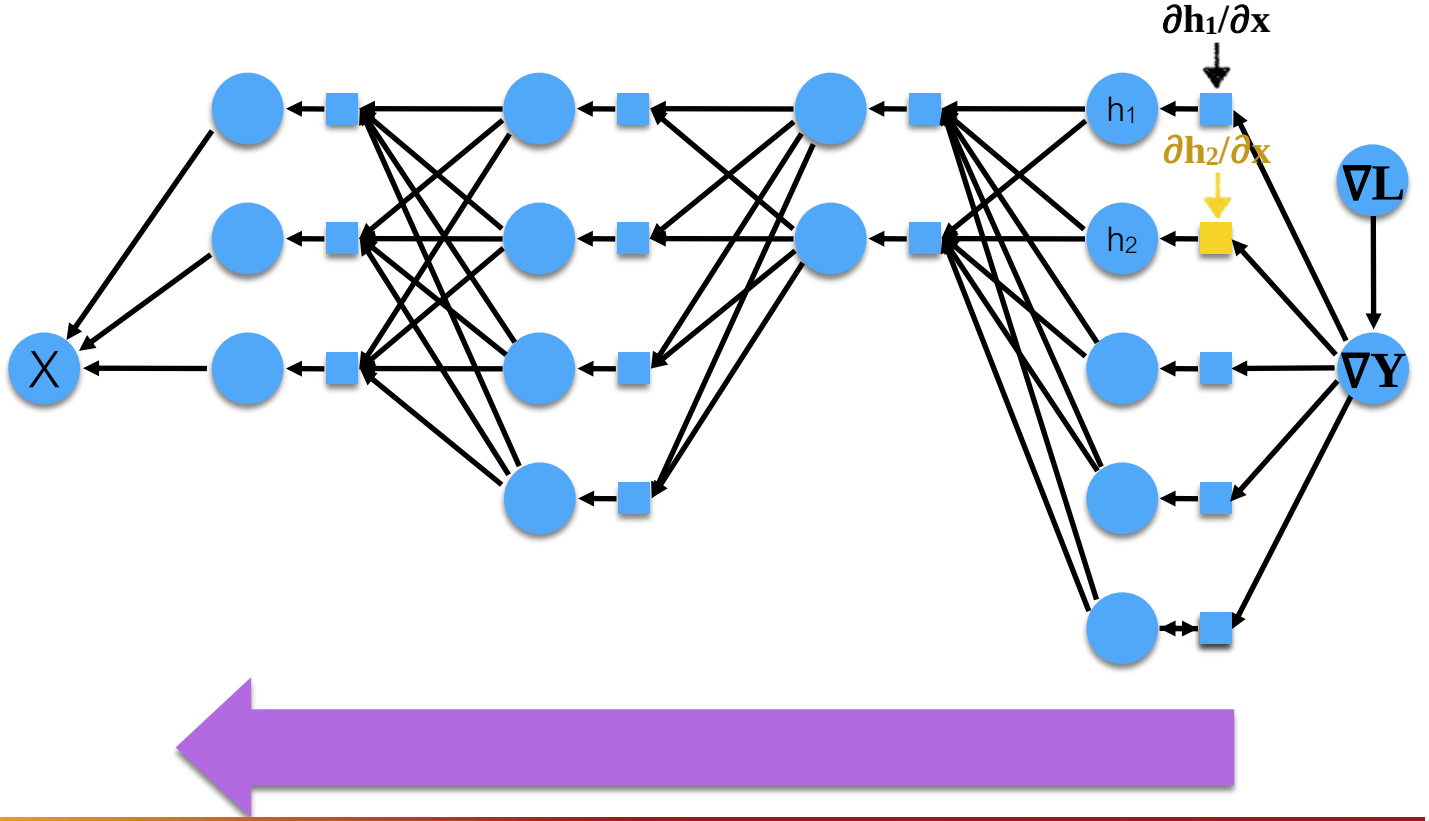
Back Propagation



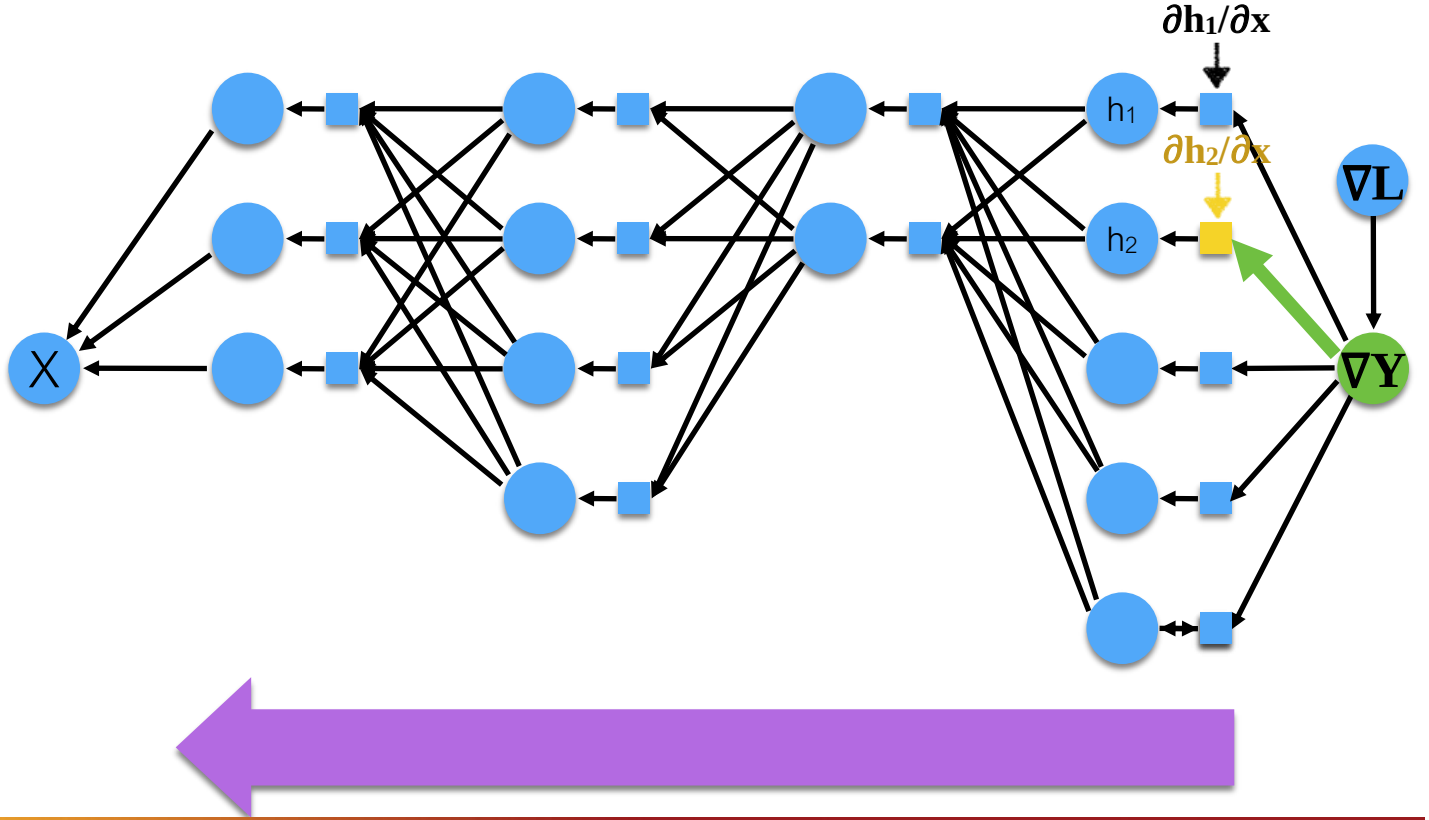
Back Propagation



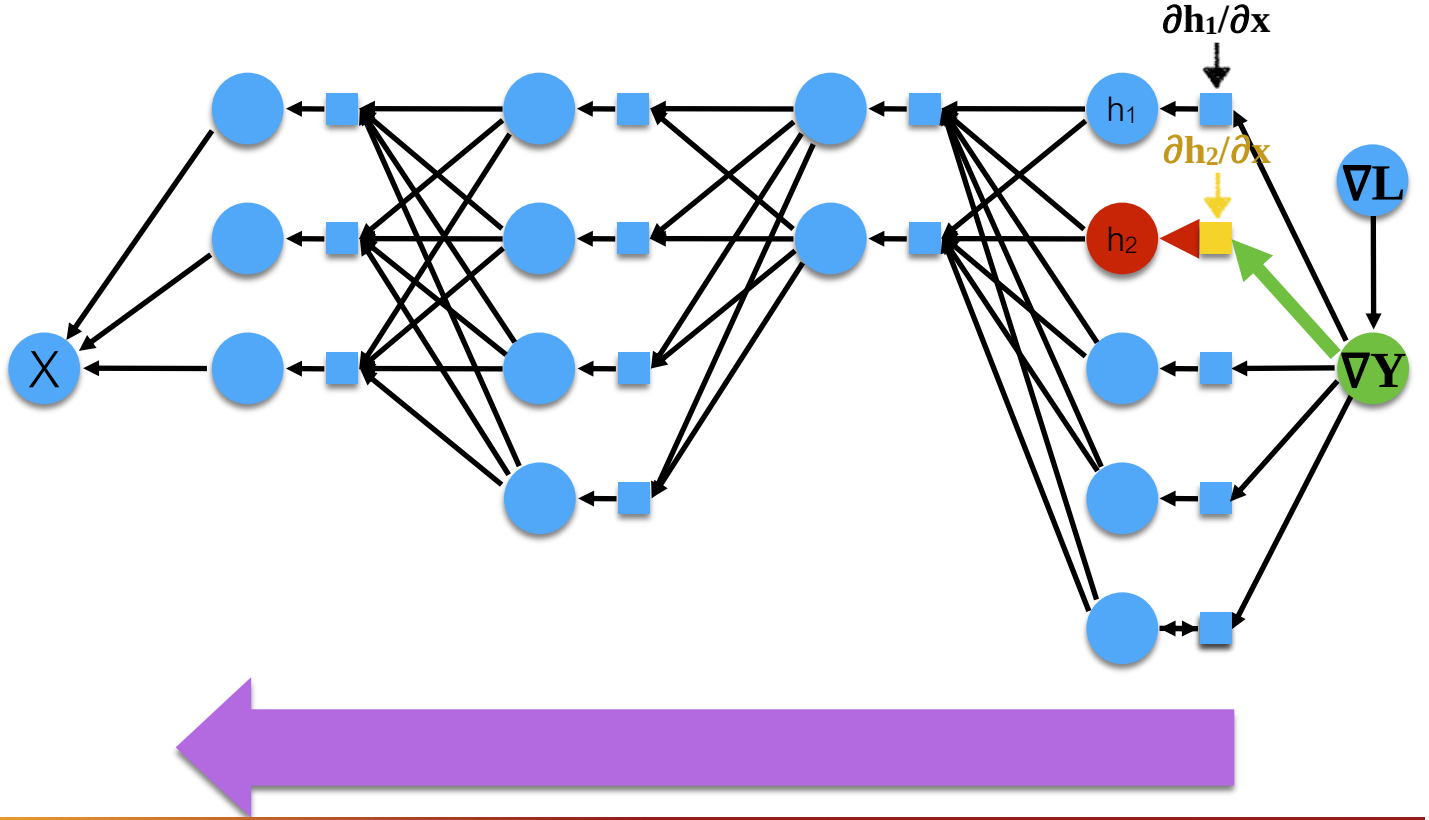
Back Propagation



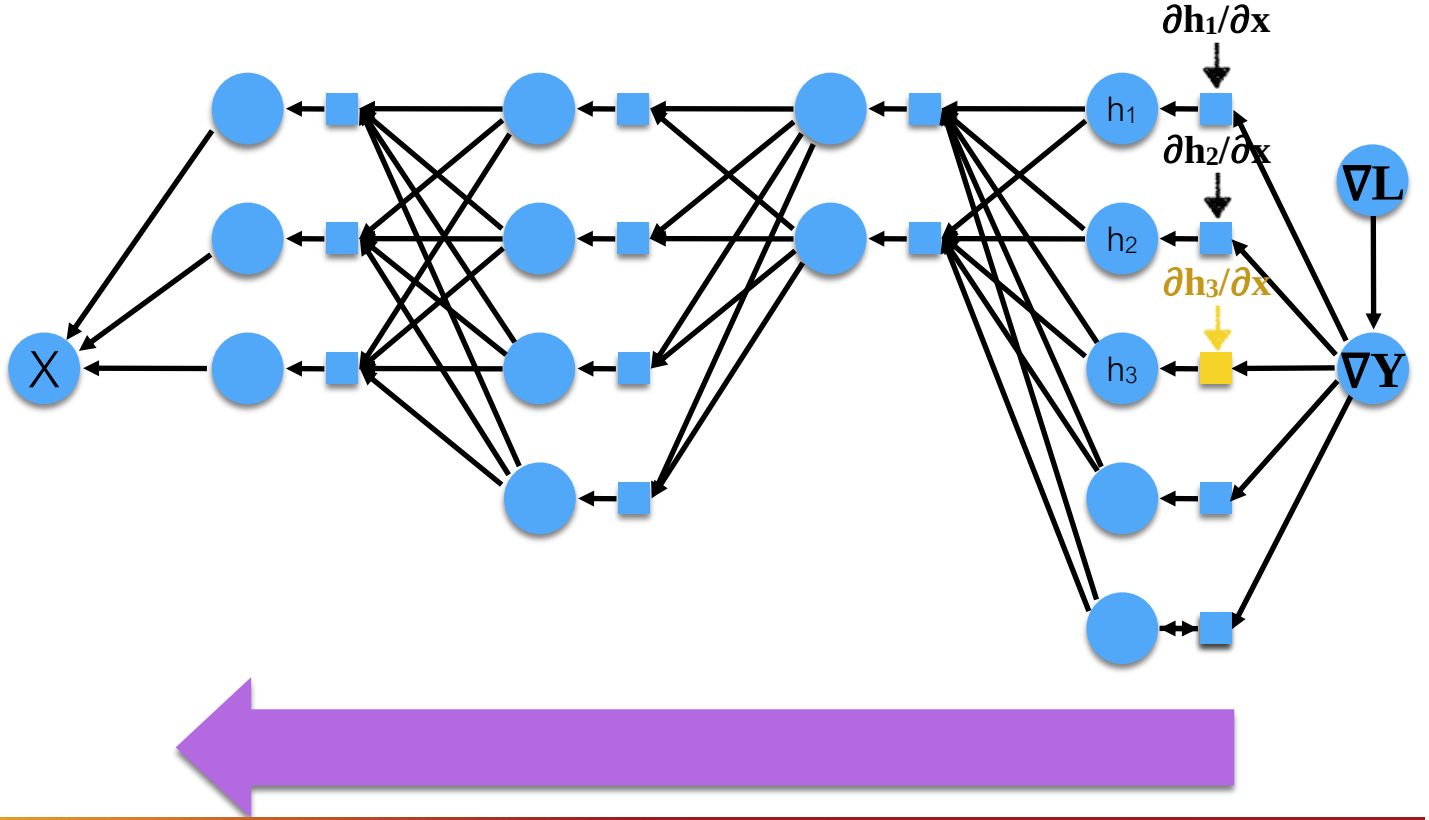
Back Propagation



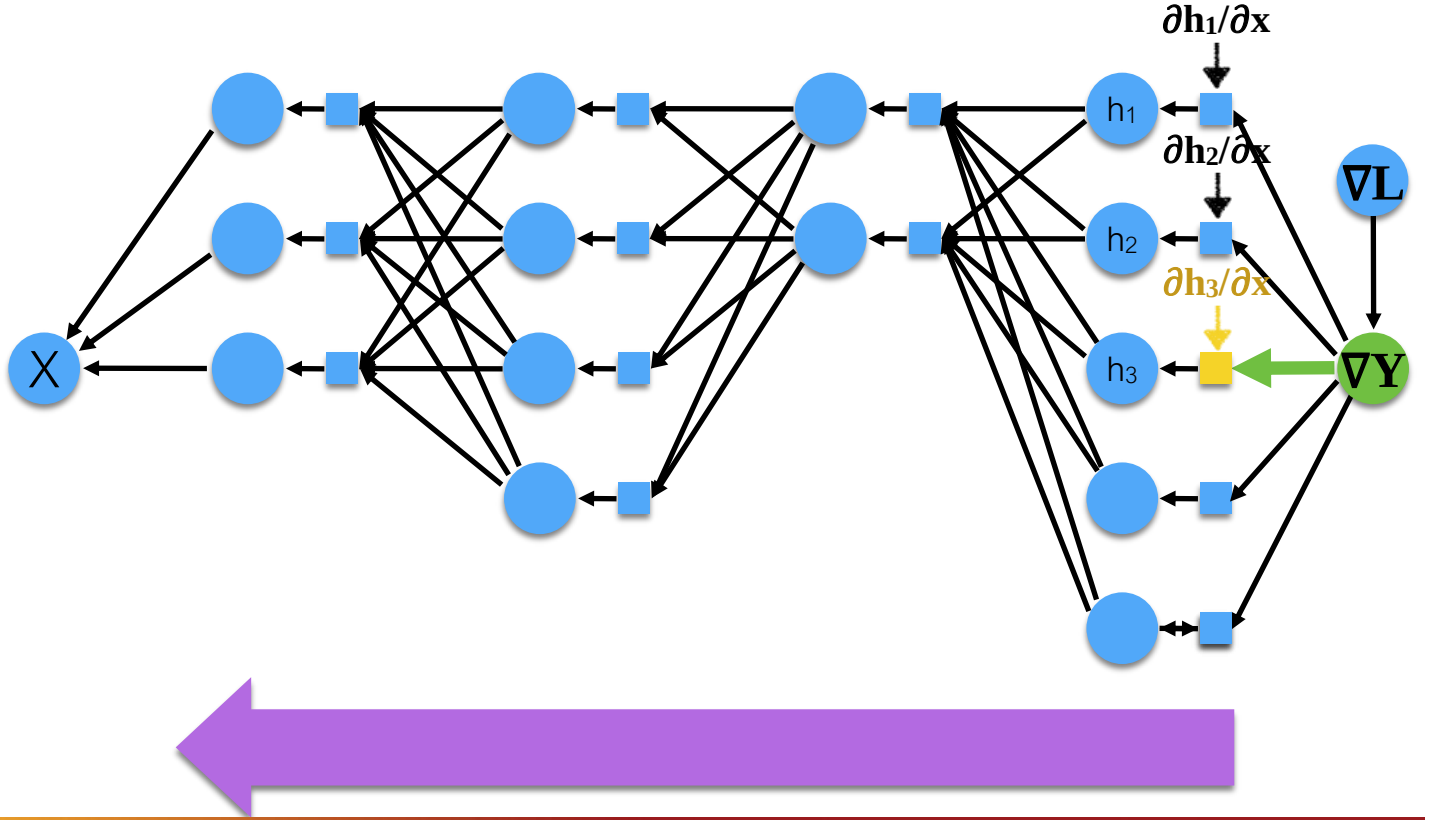
Back Propagation



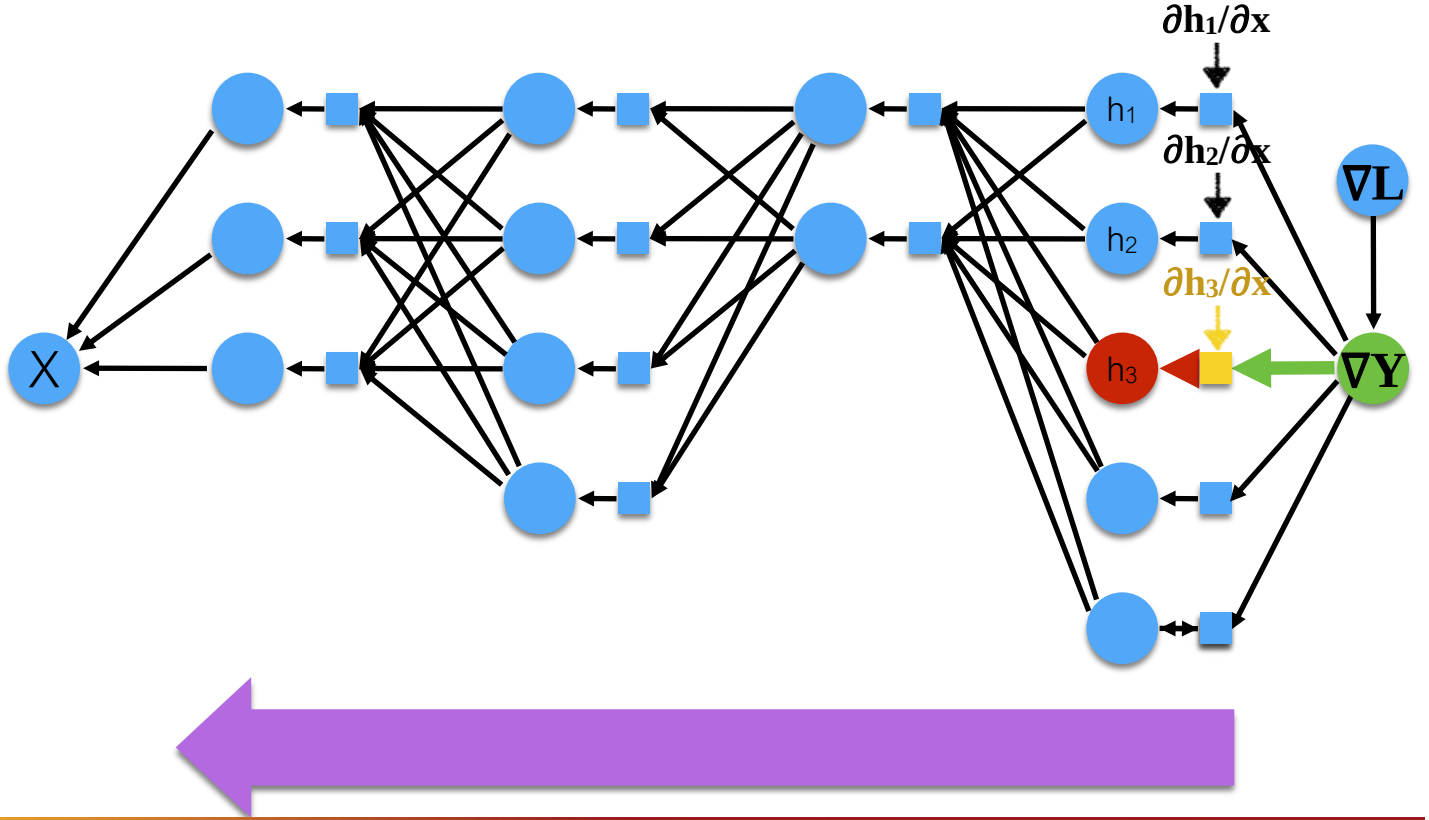
Back Propagation



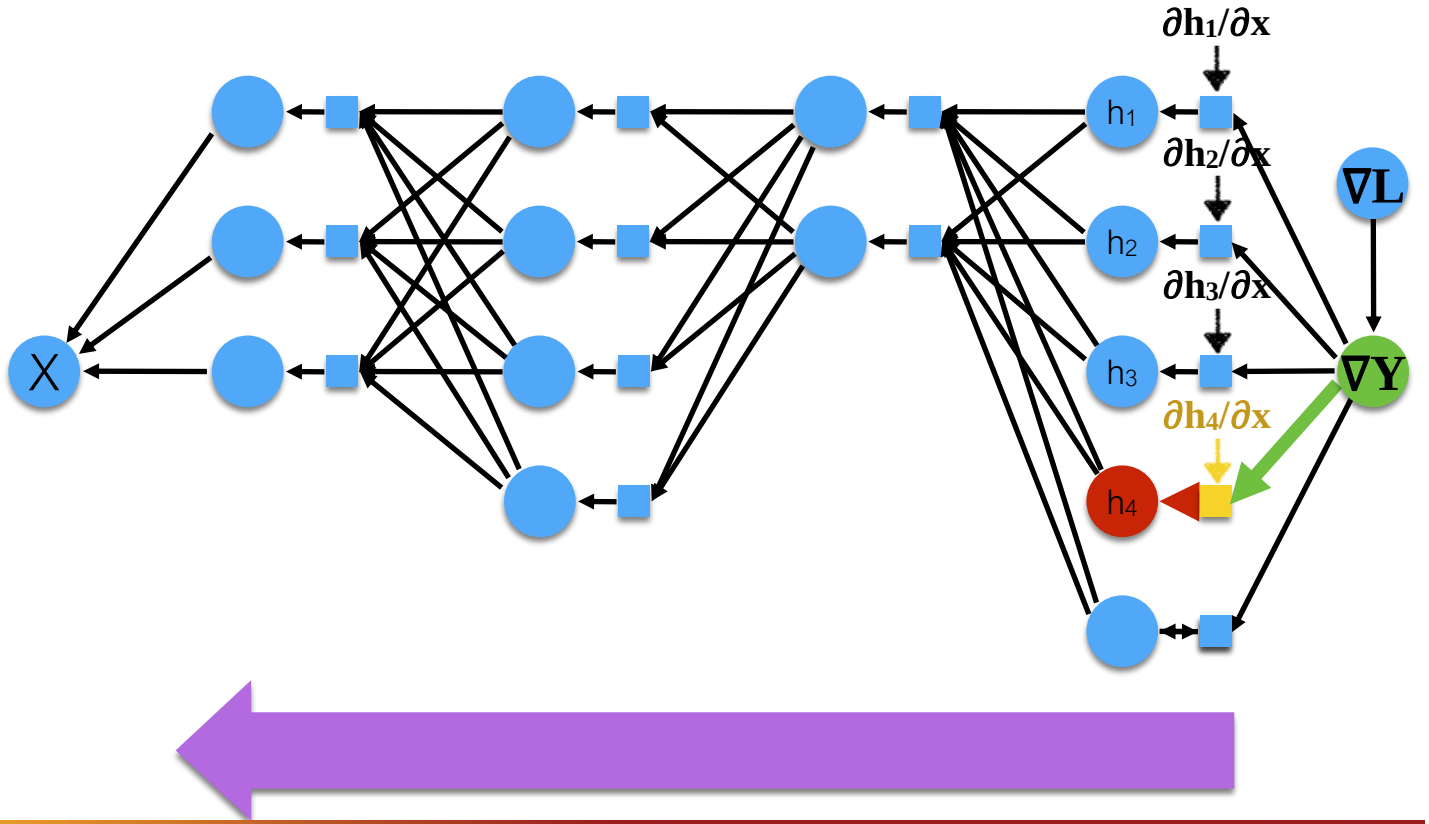
Back Propagation



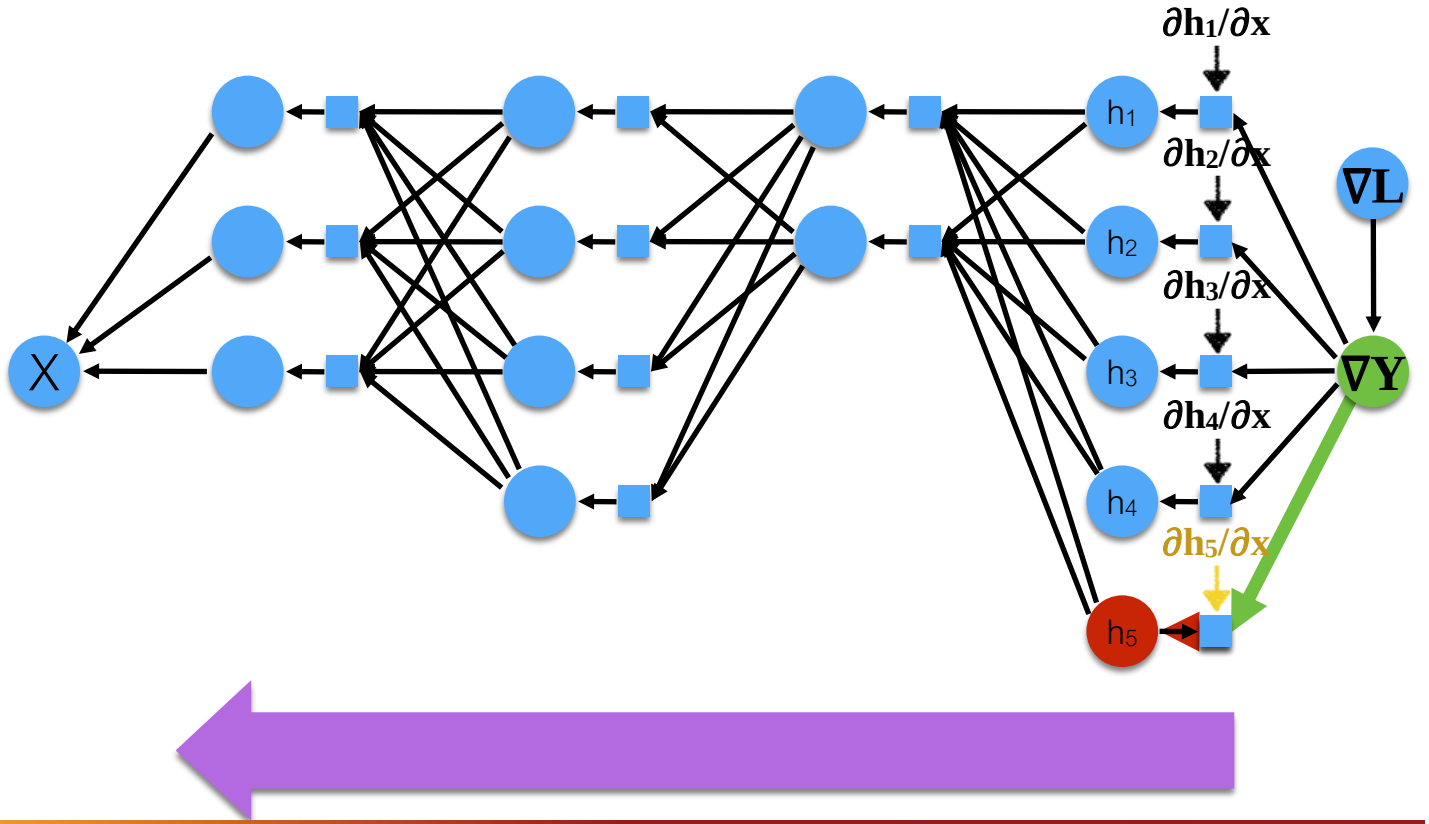
Back Propagation



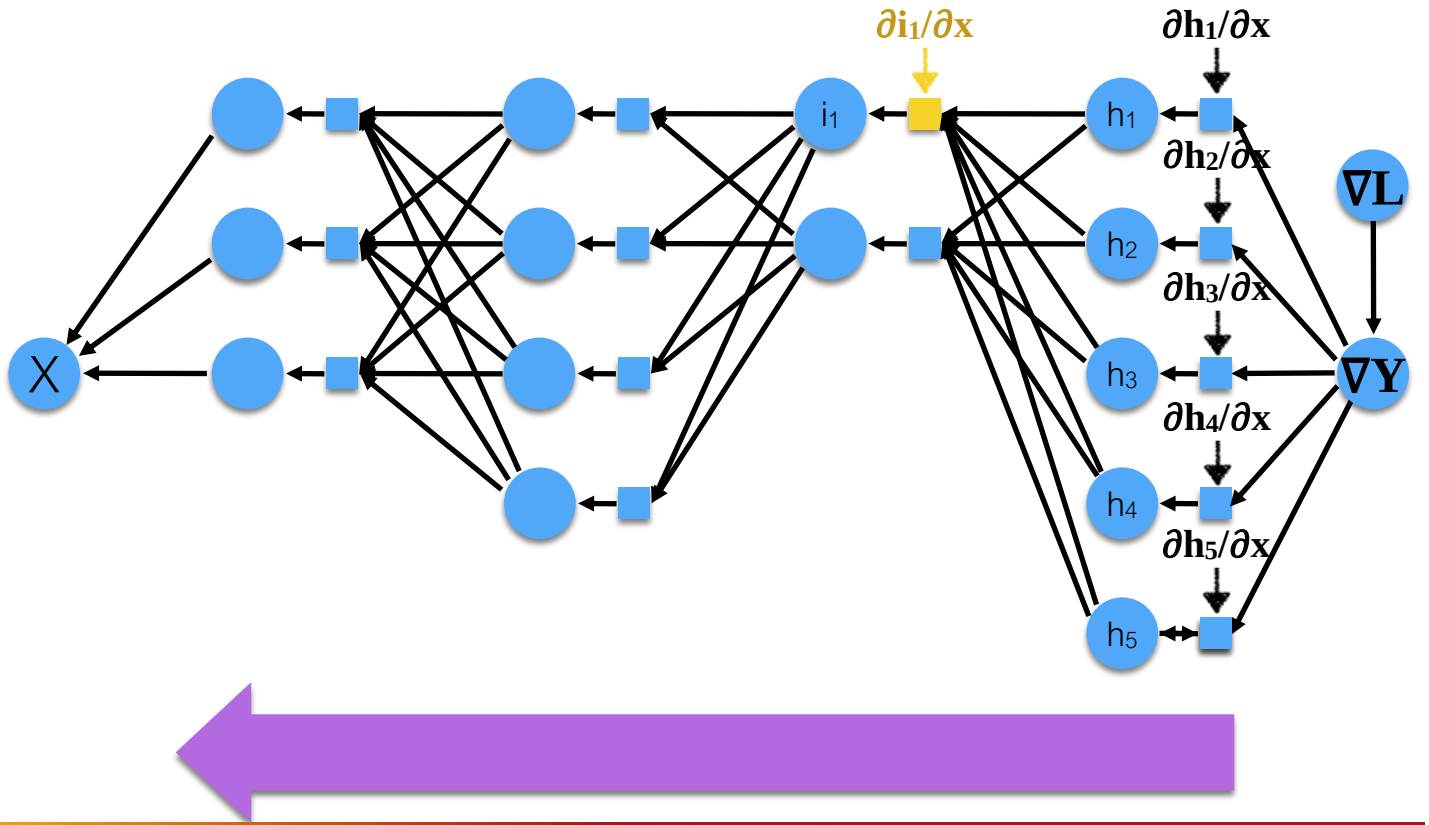
Back Propagation



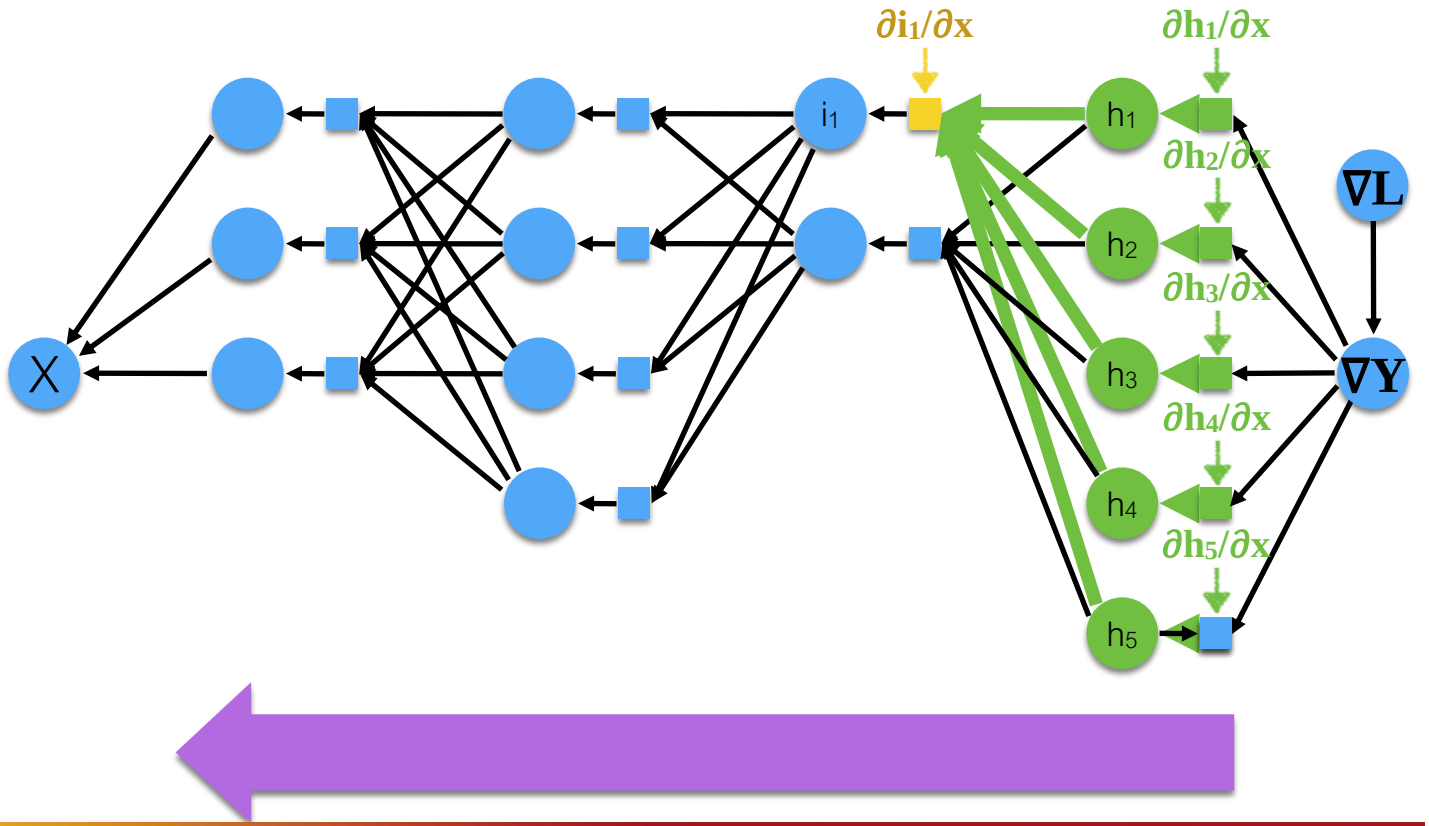
Back Propagation



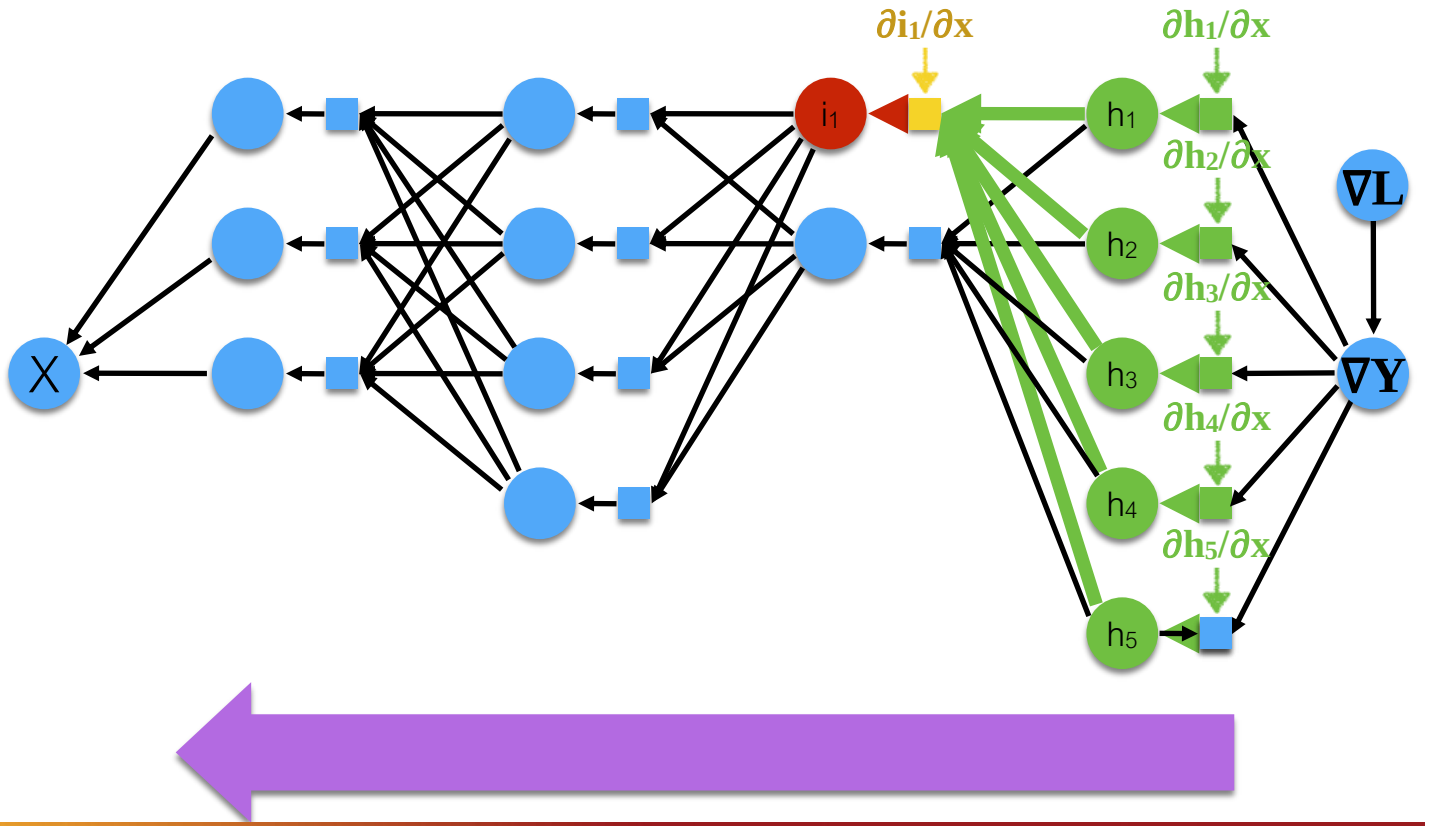
Back Propagation



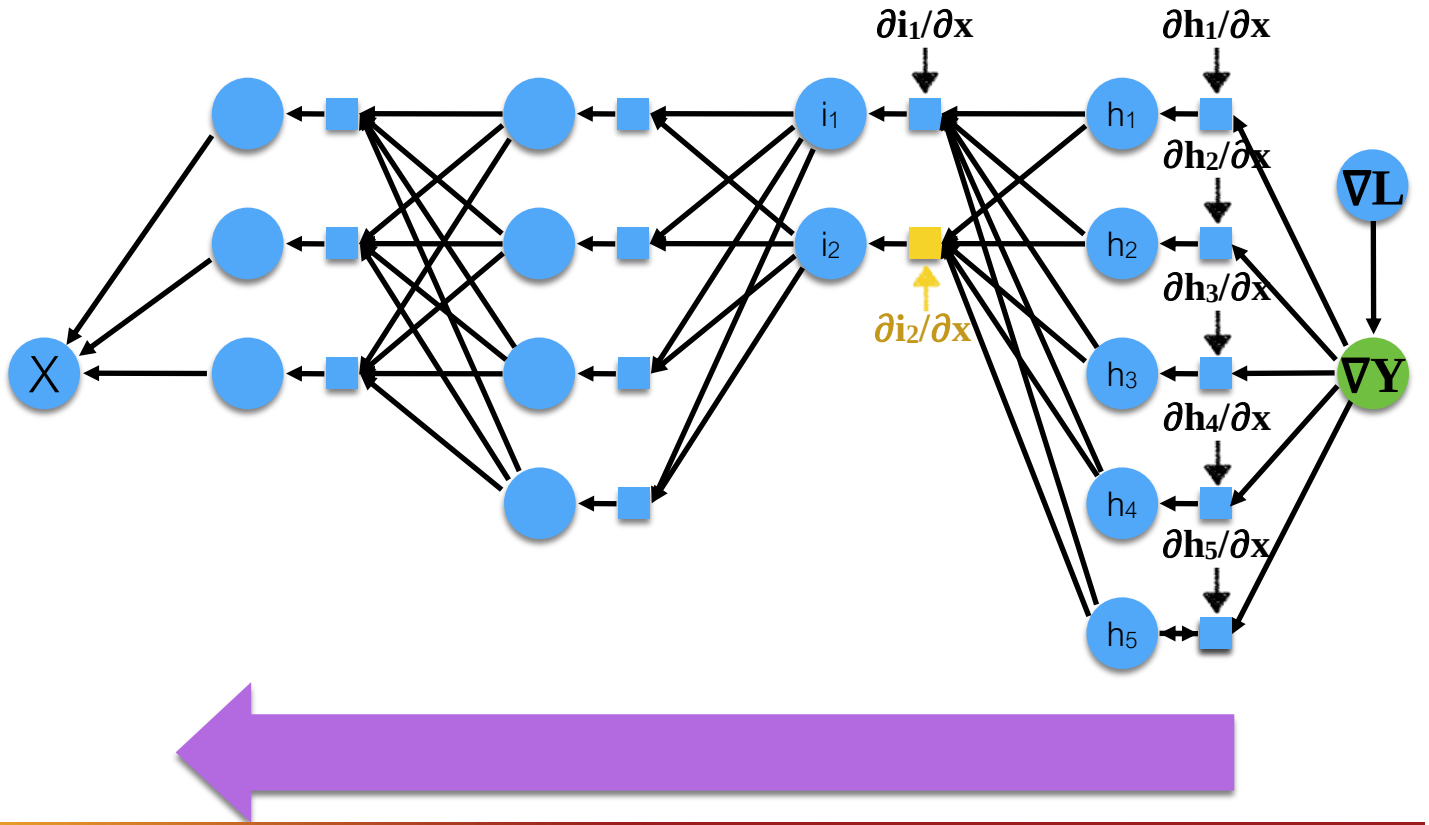
Back Propagation



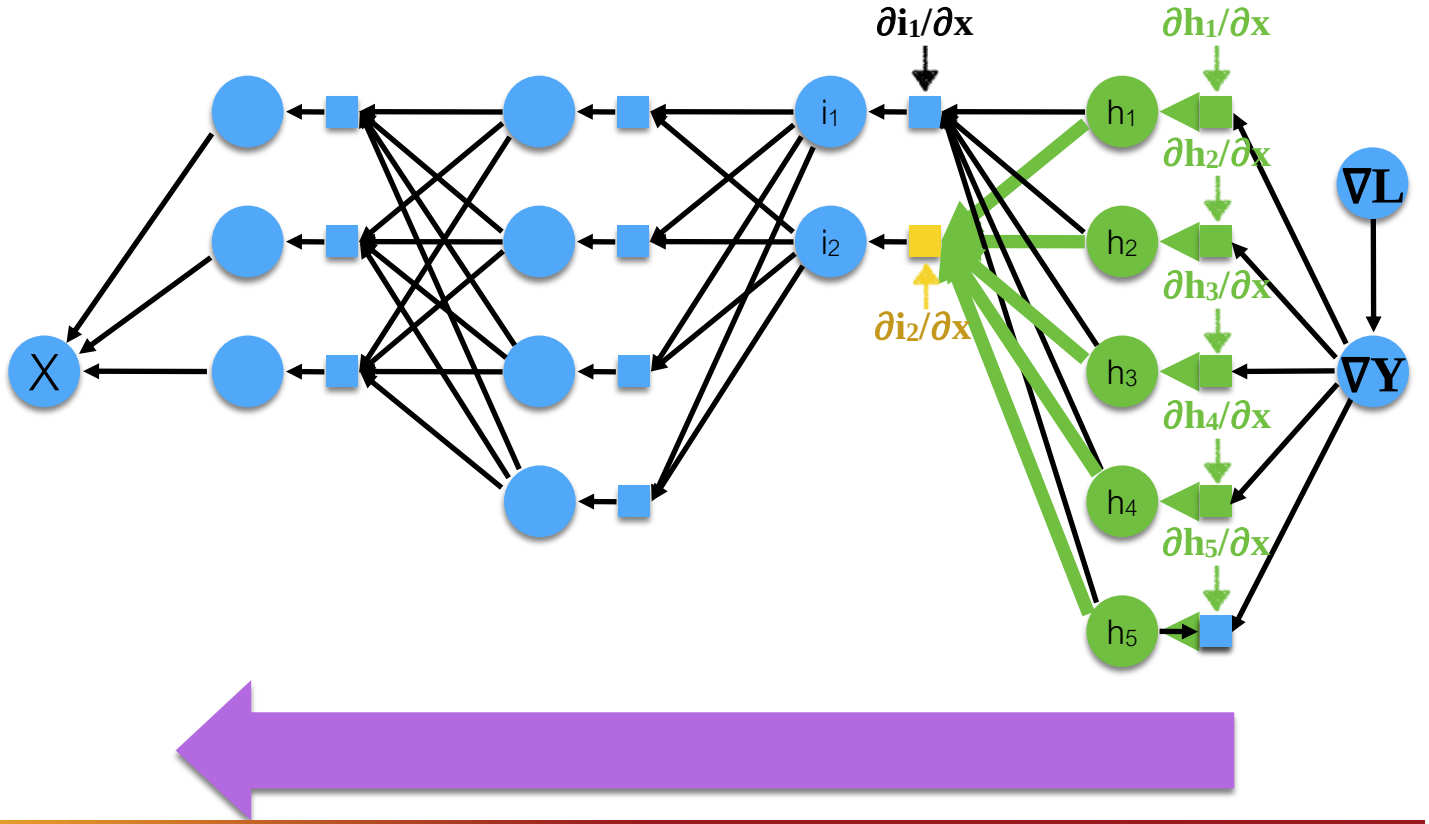
Back Propagation



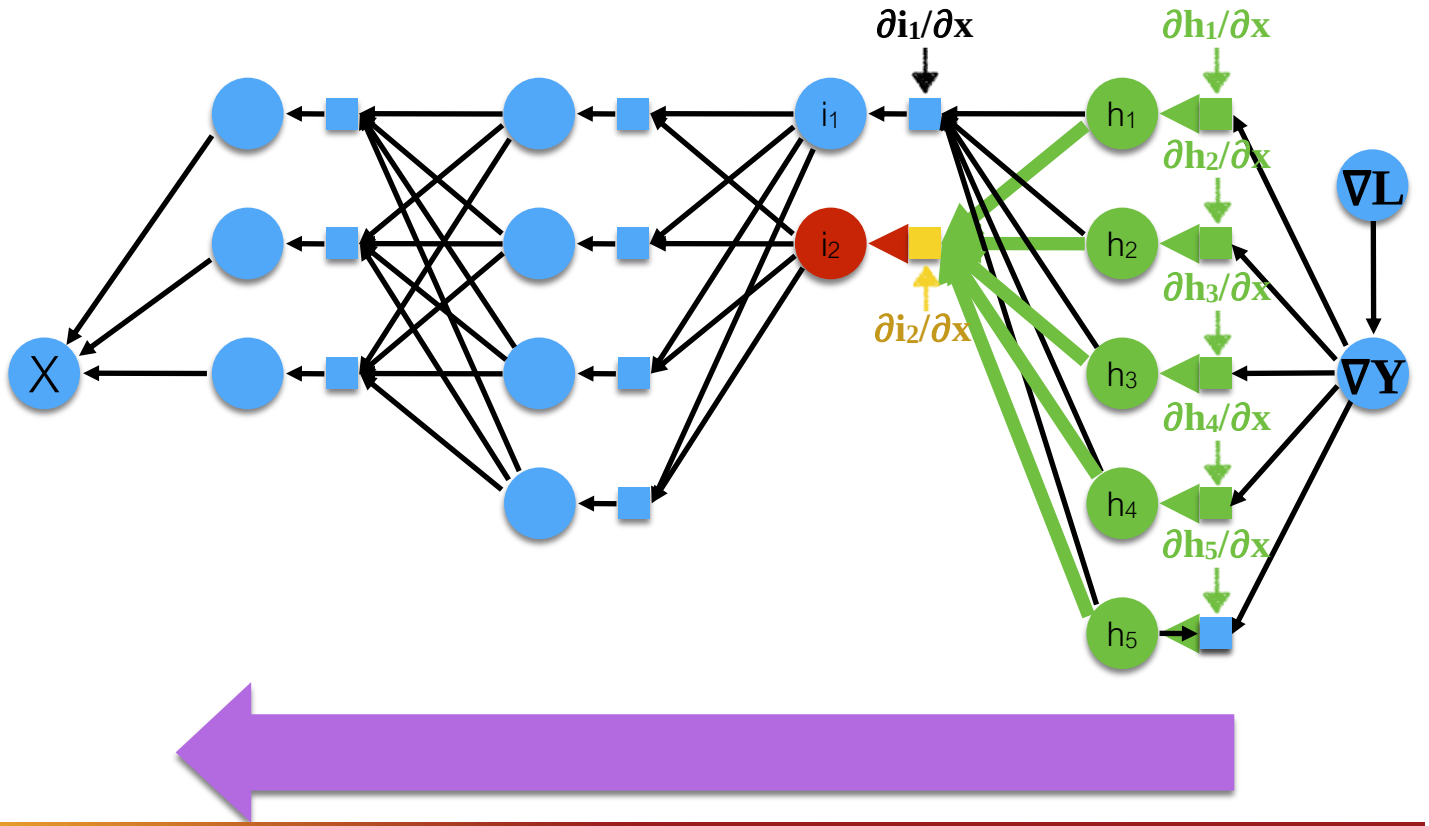
Back Propagation



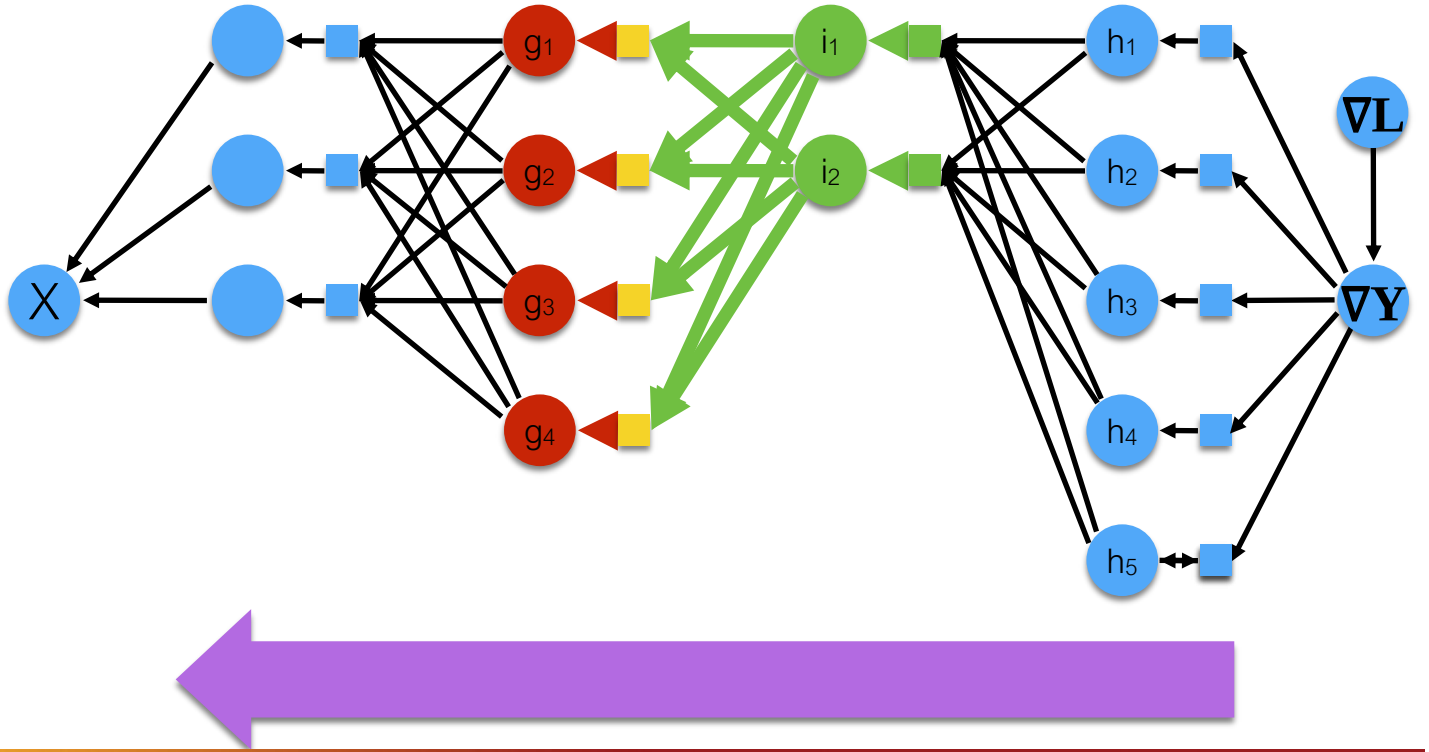
Back Propagation



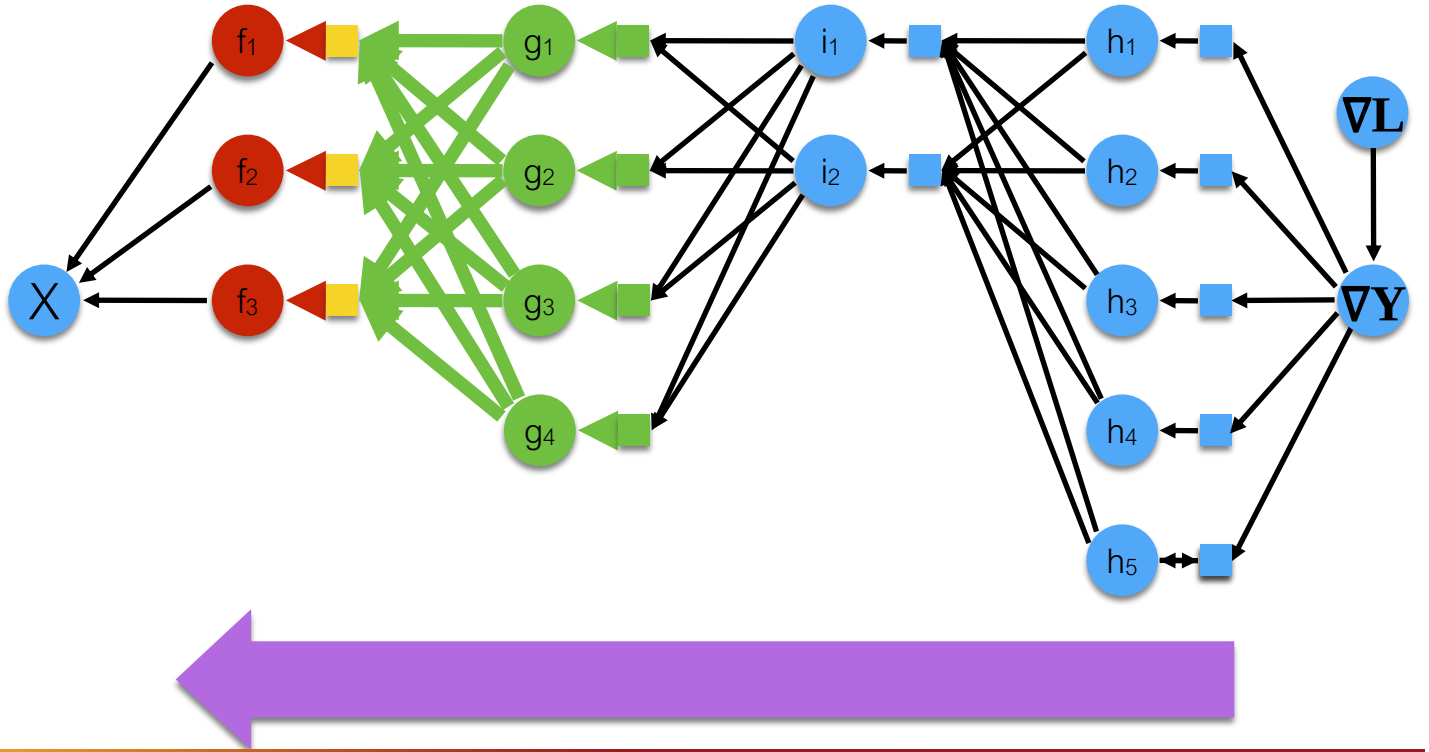
Back Propagation



Back Propagation



Back Propagation



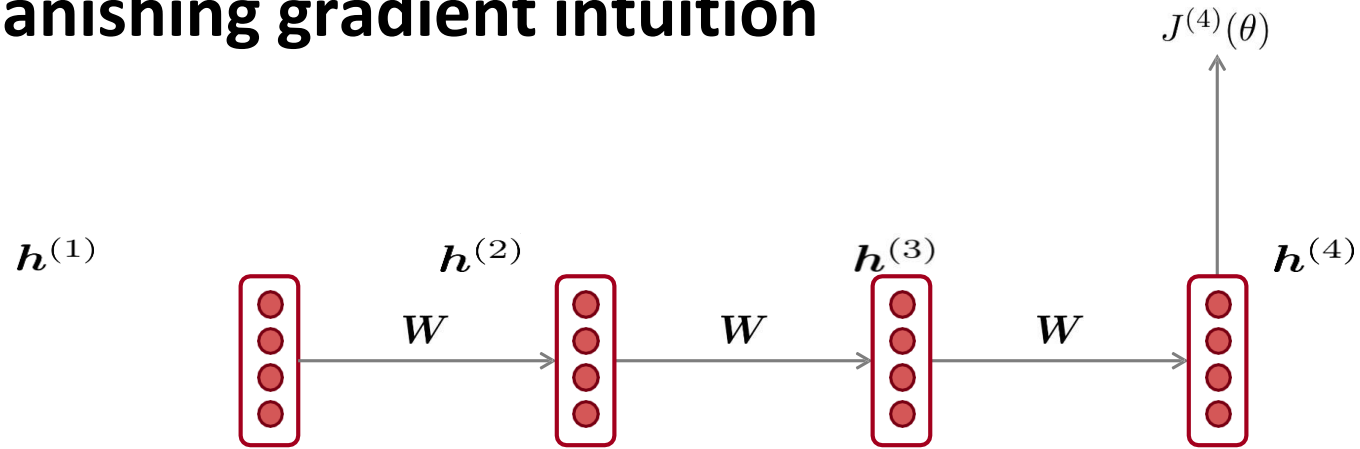
Back Propagation

- **(Fully-connected) Neural Networks** are stacks of linear functions and nonlinear activation functions; they have much more representational power than linear classifiers

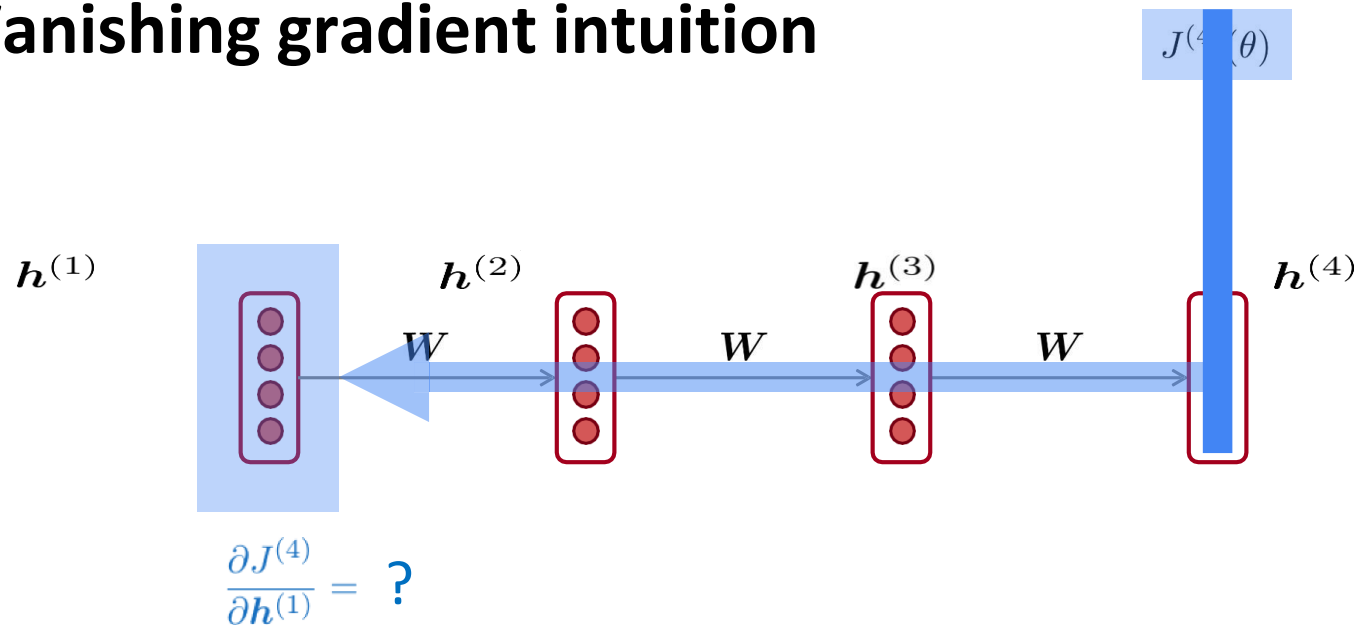
- **backpropagation** = recursive application of the chain rule along a computational graph to compute the gradients of all inputs/ parameters/ intermediates

- **forward**: compute the result of an operation and save any intermediates needed for gradient computation in memory
- **backward**: apply the chain rule to compute the gradient of the loss function with respect to the inputs

Vanishing gradient intuition

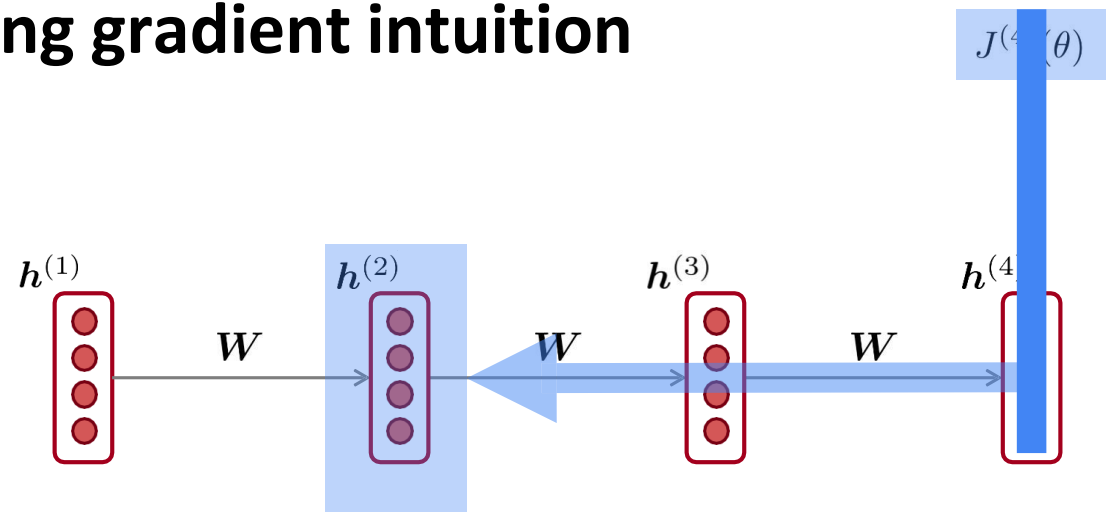


Vanishing gradient intuition



The vanishing gradient can occur in any type of deep neural network. However, it is more prominently discussed in the architecture with sequential data.

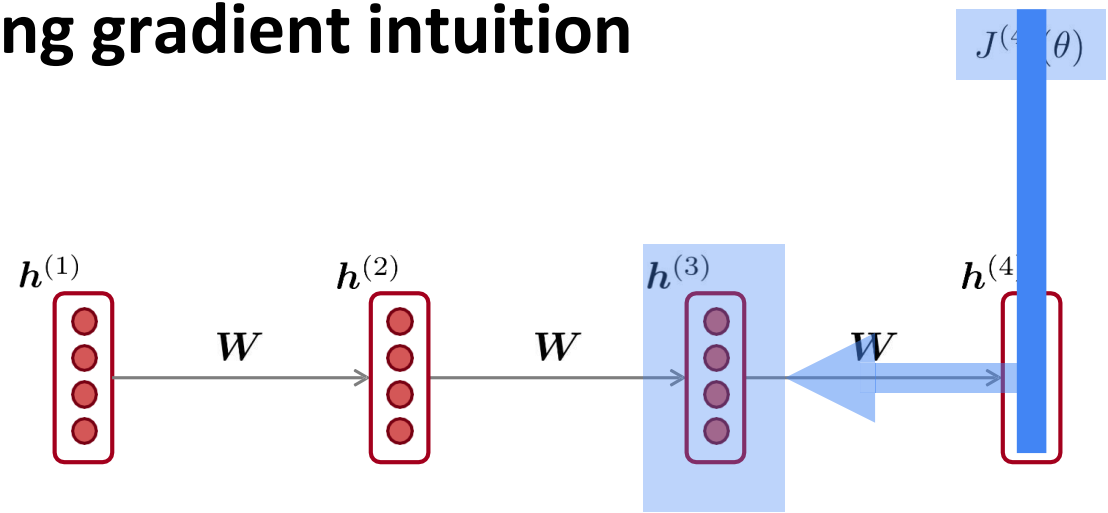
Vanishing gradient intuition



$$\frac{\partial J^{(4)}}{\partial \boldsymbol{h}^{(1)}} = \frac{\partial \boldsymbol{h}^{(2)}}{\partial \boldsymbol{h}^{(1)}} \times \frac{\partial J^{(4)}}{\partial \boldsymbol{h}^{(2)}}$$

chain rule!

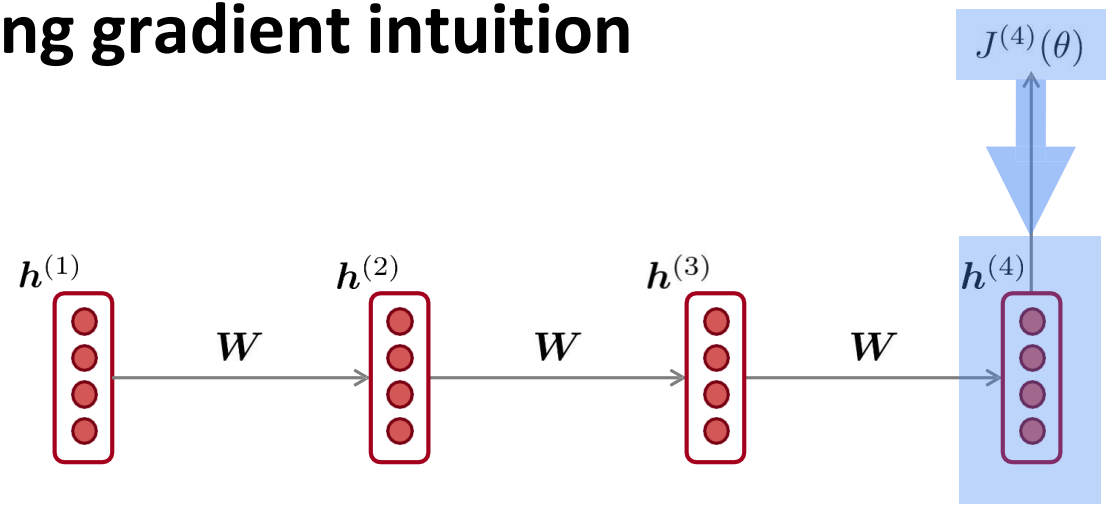
Vanishing gradient intuition



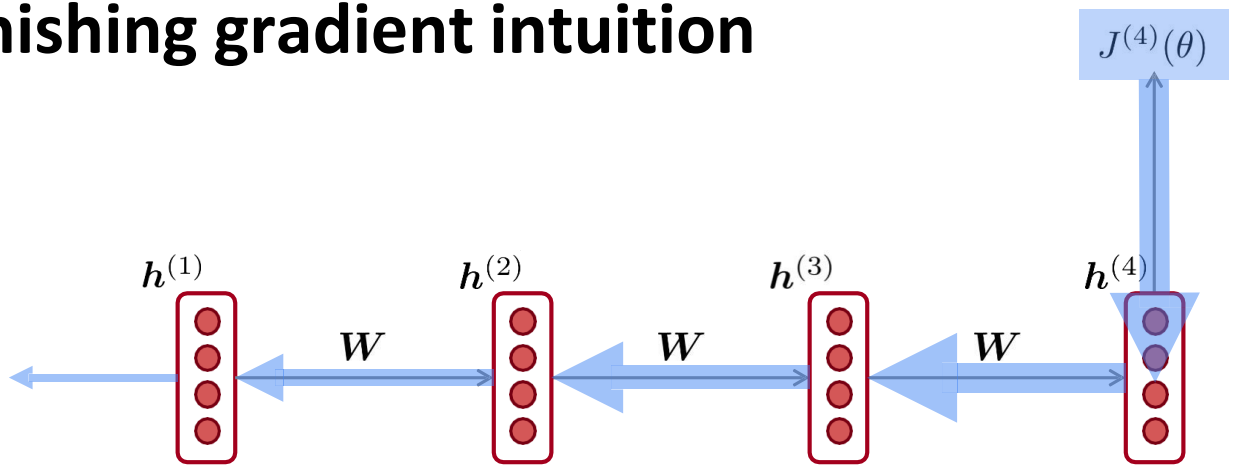
$$\frac{\partial J^{(4)}}{\partial h^{(1)}} = \frac{\partial h^{(2)}}{\partial h^{(1)}} \times \frac{\partial h^{(3)}}{\partial h^{(2)}} \times \frac{\partial J^{(4)}}{\partial h^{(3)}}$$

chain rule!

Vanishing gradient intuition



Vanishing gradient intuition

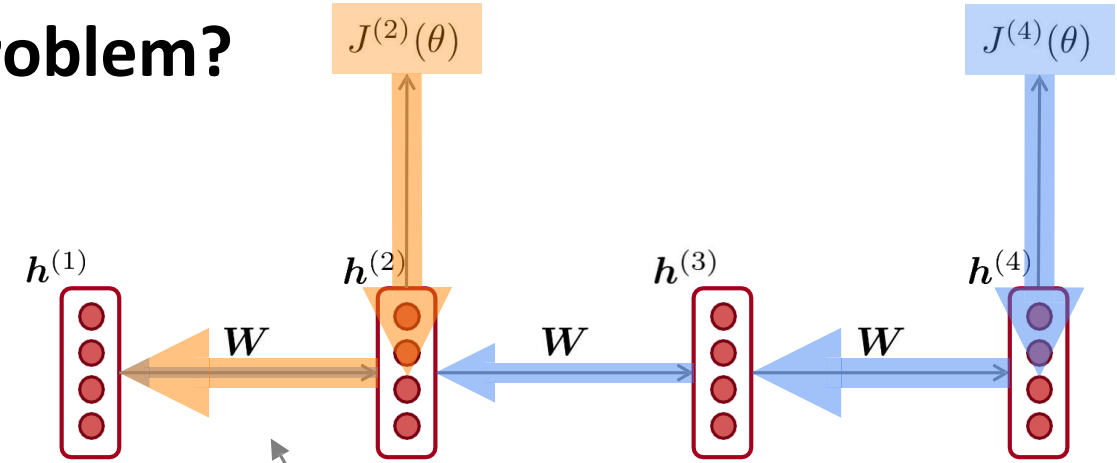


$$\frac{\partial J^{(4)}}{\partial h^{(1)}} = \frac{\partial h^{(2)}}{\partial h^{(1)}} \times \frac{\partial h^{(3)}}{\partial h^{(2)}} \times \frac{\partial h^{(4)}}{\partial h^{(3)}} \times \frac{\partial J^{(4)}}{\partial h^{(4)}}$$

What happens if these are small?

Vanishing gradient problem:
When these are small, the gradient signal gets smaller and smaller as it backpropagates further

Why is vanishing gradient a problem?



Gradient signal from faraway is lost because it's much smaller than gradient signal from close-by.

So model weights are only updated only with respect to near effects, not long-term effects.

Why is vanishing gradient a problem?

- Another explanation: Gradient can be viewed as a measure of *the effect of the past on the future*

- If the gradient becomes vanishingly small over longer distances (step *t* to step *t+n*), then we can't tell whether:
  1. There's no dependency between step *t* and *t+n* in the data
  2. We have wrong parameters to capture the true dependency between *t* and *t+n*

Fixing Vanishing Gradients

Use of Appropriate Activation Functions:

- ReLU and its Variants: ReLU (Rectified Linear Unit) is less susceptible to the vanishing gradient problem because it does not saturate in the positive region. Variants like Leaky ReLU, Parametric ReLU (PReLU), or Exponential Linear Unit (ELU) can also be used to avoid dead neurons in ReLU.

Batch Normalization (will discuss later – the idea is to fix the range):

- This technique normalizes the input of each layer to have mean 0 and variance 1. This normalization helps to combat the vanishing gradient problem by reducing internal covariate shift.

Why is exploding gradient a problem?

- If the gradient becomes too big, then the SGD update step becomes too big:

$$\theta^{new} = \theta^{old} - \overbrace{\alpha}^{\text{learning rate}} \underbrace{\nabla_\theta J(\theta)}_{\text{gradient}}$$

- This can cause bad updates: we take too large a step and reach a bad parameter configuration (with large loss)

- In the worst case, this will result in Inf or NaN in your network (then you have to restart training from an earlier checkpoint)

Gradient clipping: solution for exploding gradient

- Gradient clipping: if the norm of the gradient is greater than some threshold, scale it down before applying SGD update

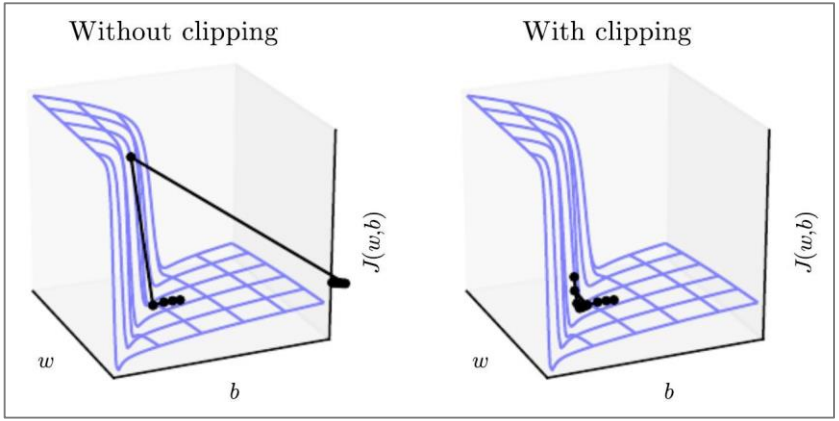
Algorithm 1 Pseudo-code for norm clipping

$\hat{\mathbf{g}} \leftarrow \frac{\partial \mathcal{E}}{\partial \theta}$
if $\|\hat{\mathbf{g}}\| \geq threshold$ **then**
  $\hat{\mathbf{g}} \leftarrow \frac{threshold}{\|\hat{\mathbf{g}}\|} \hat{\mathbf{g}}$
end if

- Intuition: take a step in the same direction, but a smaller step

Source: "On the difficulty of training recurrent neural networks", Pascanu et al, 2013. http://proceedings.mlr.press/v28/pascanu13.pdf

Gradient clipping: solution for exploding gradient



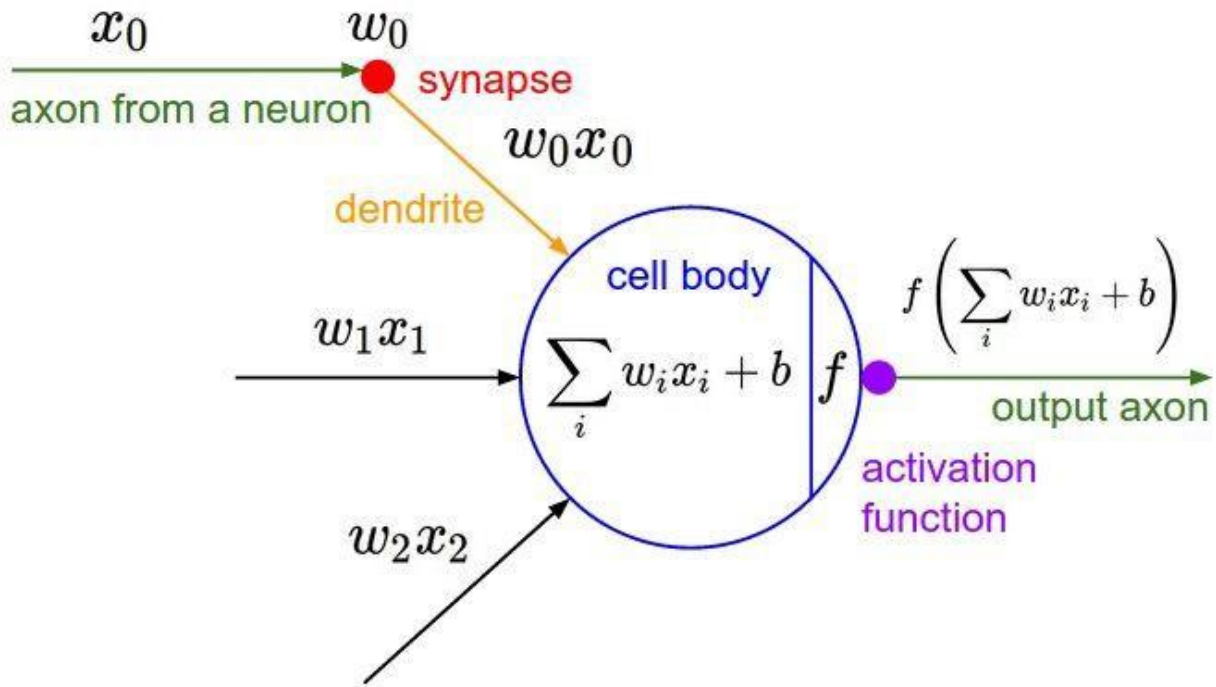
- This shows the loss surface of a simple RNN (hidden state is a scalar not a vector)
- The "cliff" is dangerous because it has steep gradient
- On the left, gradient descent takes two very big steps due to steep gradient, resulting in climbing the cliff then shooting off to the right (both bad updates)
- On the right, gradient clipping reduces the size of those steps, so effect is less drastic

Source: "Deep Learning", Goodfellow, Bengio and Courville, 2016. Chapter 10.11.1. https://www.deeplearningbook.org/contents/rnn.html

Training Neural Networks

Activation Functions

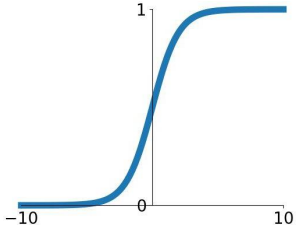
Activation Functions



Activation Functions

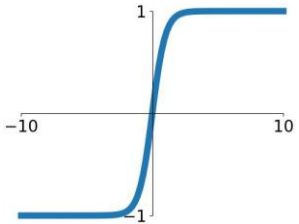
Sigmoid

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



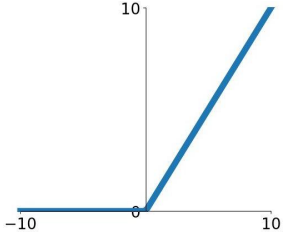
tanh

$$\tanh(x)$$



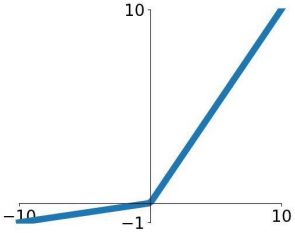
ReLU

$$\max(0, x)$$



Leaky ReLU

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

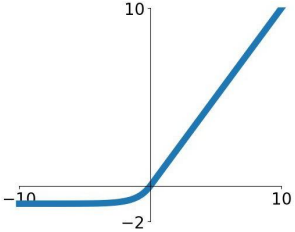


Maxout

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

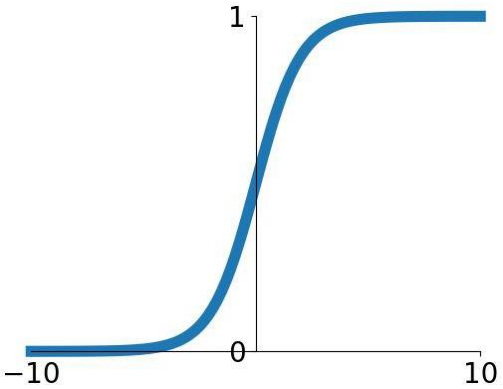
ELU

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



Activation Functions

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

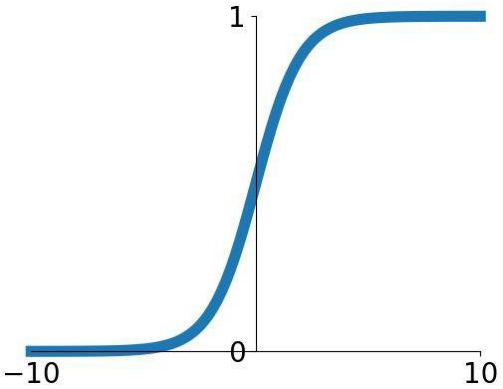


Sigmoid

- Squashes numbers to range [0,1]
- Historically popular since they have nice interpretation as a saturating "firing rate" of a neuron

Activation Functions

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

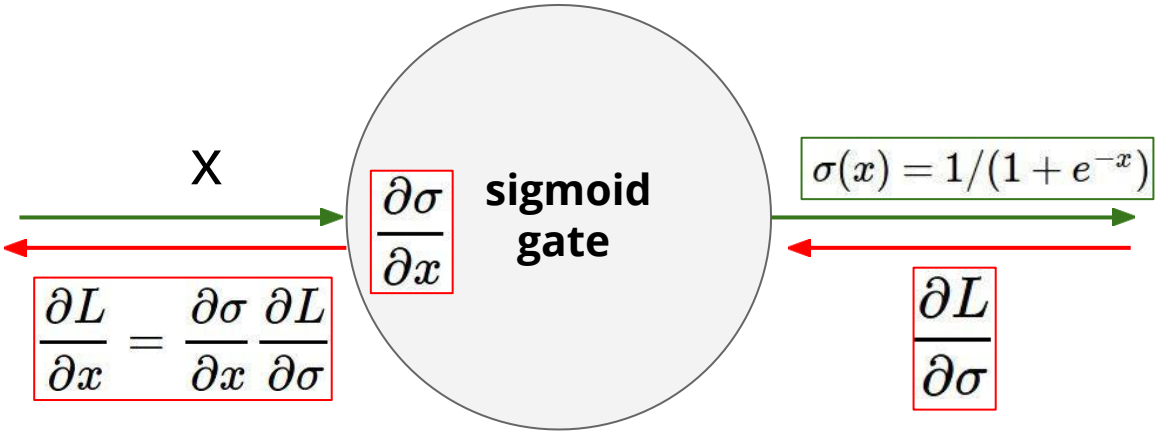


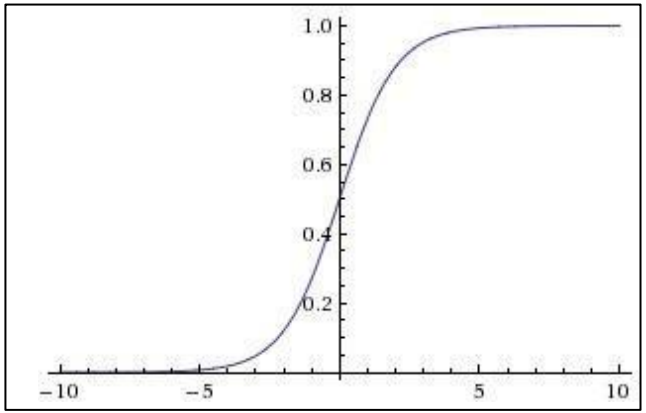
Sigmoid

- Squashes numbers to range [0,1]
- Historically popular since they have nice interpretation as a saturating "firing rate" of a neuron

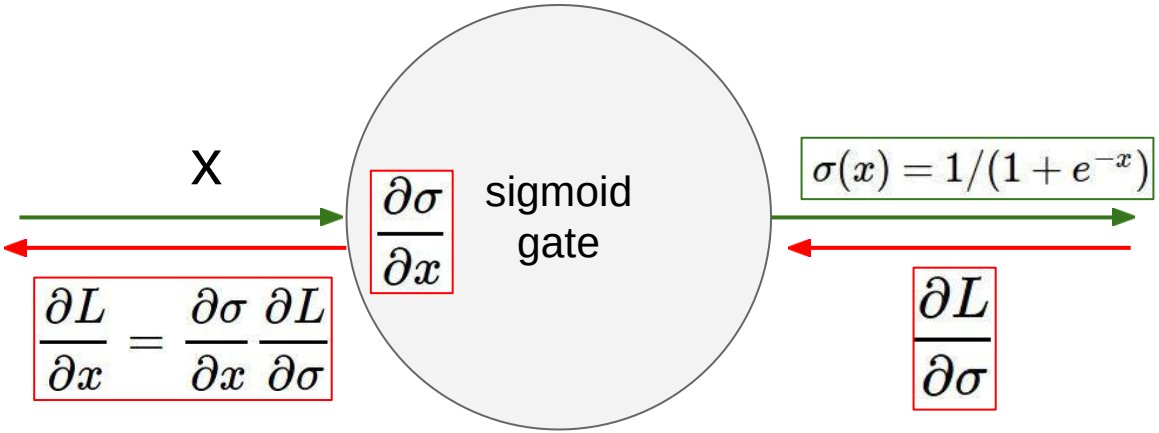
3 problems:

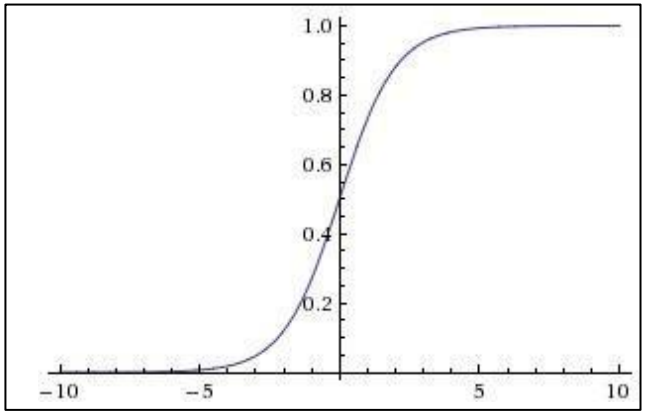
1. Saturated neurons "kill" the gradients





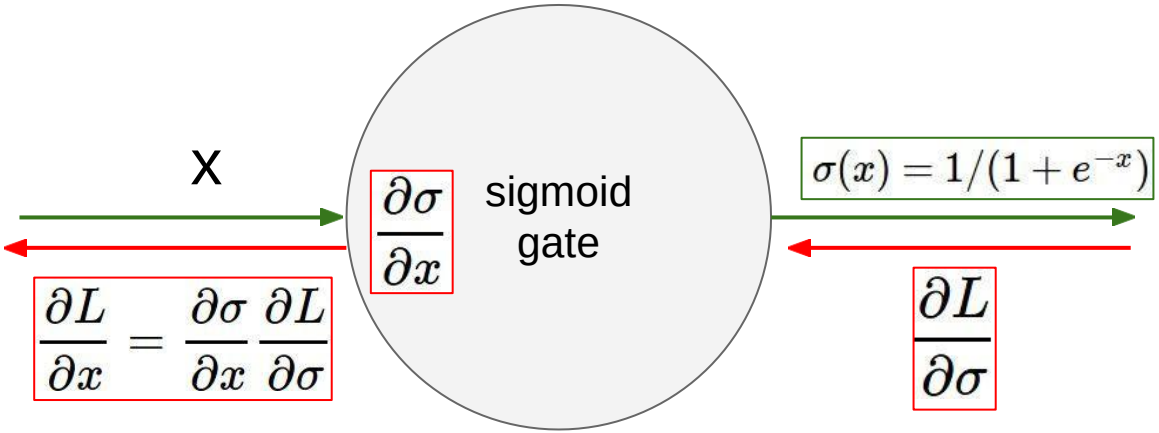
$$\frac{\partial \sigma(x)}{\partial x} = \sigma(x)\,(1 - \sigma(x))$$

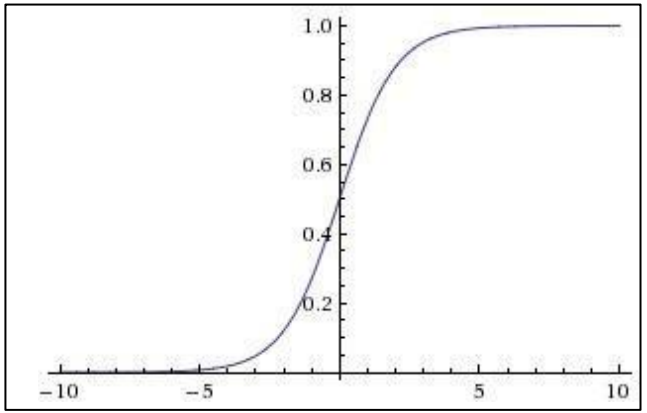




What happens when x = -10?

$$\frac{\partial \sigma(x)}{\partial x} = \sigma(x)\,(1 - \sigma(x))$$



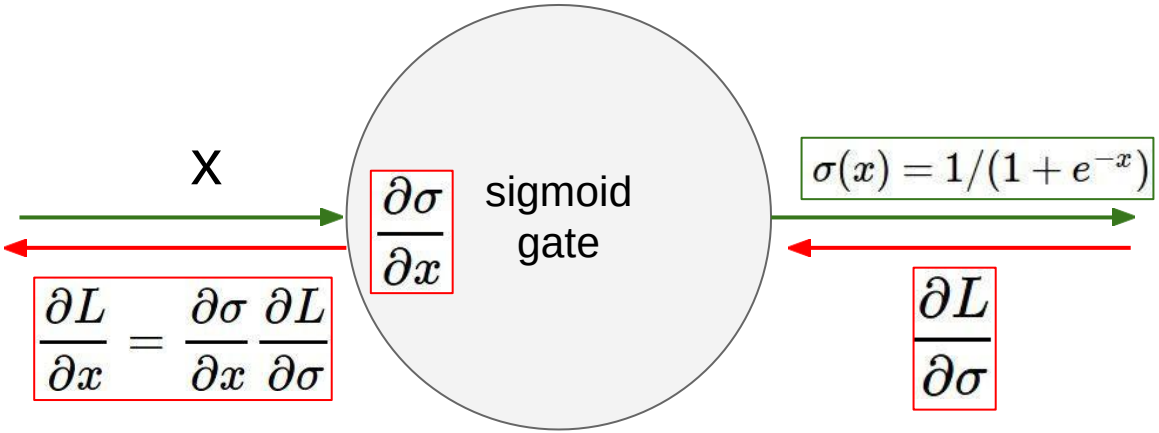


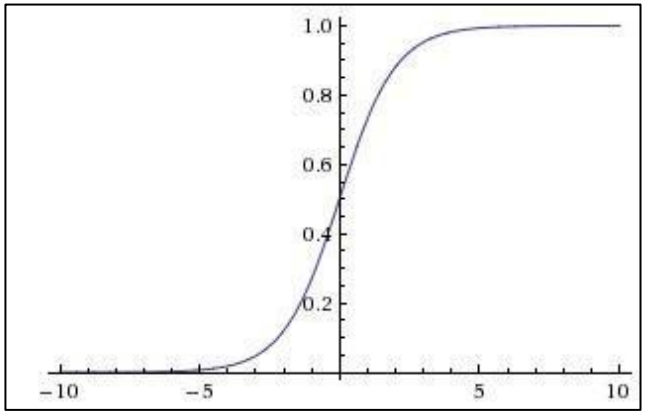
What happens when x = -10?

$$\frac{\partial \sigma(x)}{\partial x} = \sigma(x)\,(1-\sigma(x))$$

$$\sigma(x) = \sim 0$$

$$\frac{\partial \sigma(x)}{\partial x} = \sigma(x)\,(1-\sigma(x)) = 0(1-0) = 0$$

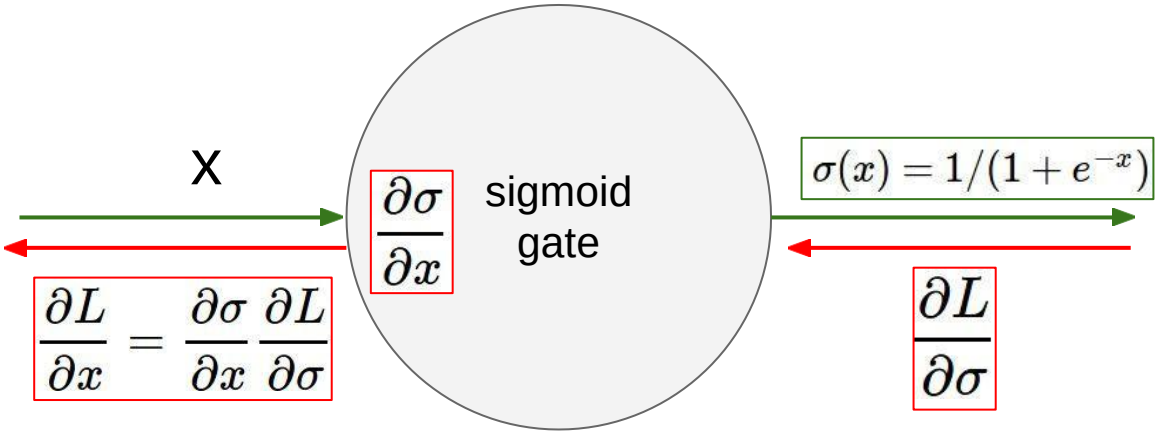


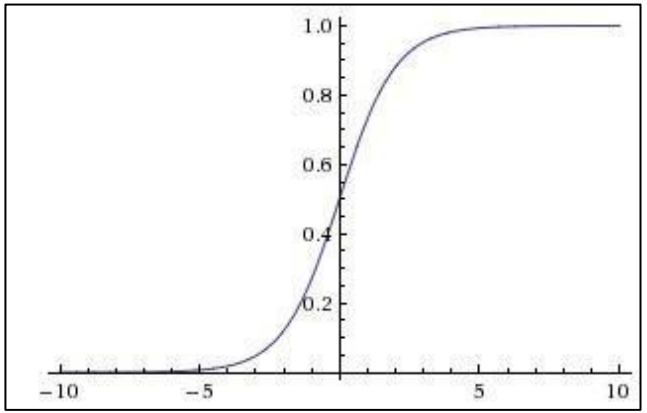


What happens when x = -10?

What happens when x = 0?

$$\frac{\partial \sigma(x)}{\partial x} = \sigma(x)\,(1 - \sigma(x))$$





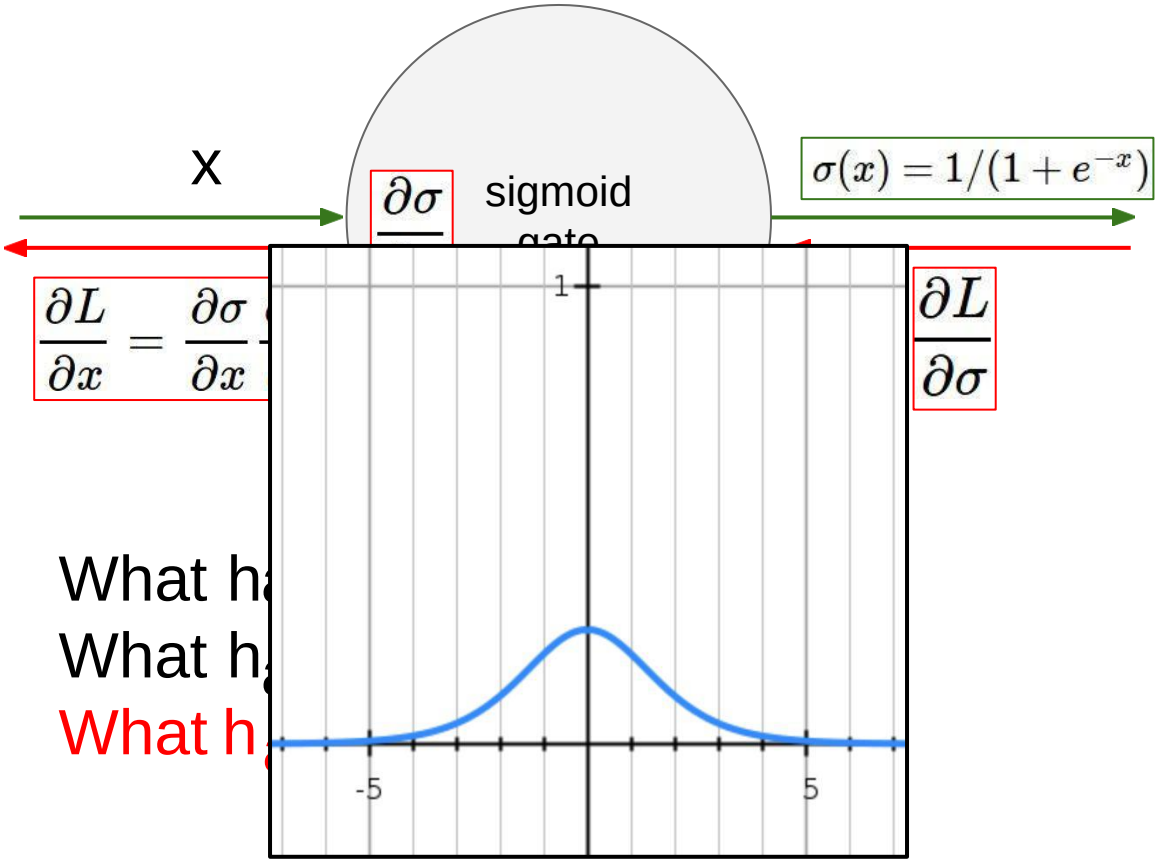
What happens when x = -10?

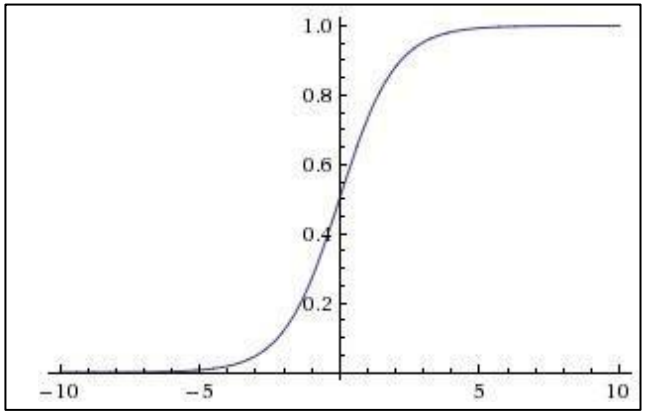
What happens when x = 0?

What happens when x = 10?

$$\frac{\partial \sigma(x)}{\partial x} = \sigma(x)\,(1-\sigma(x))$$

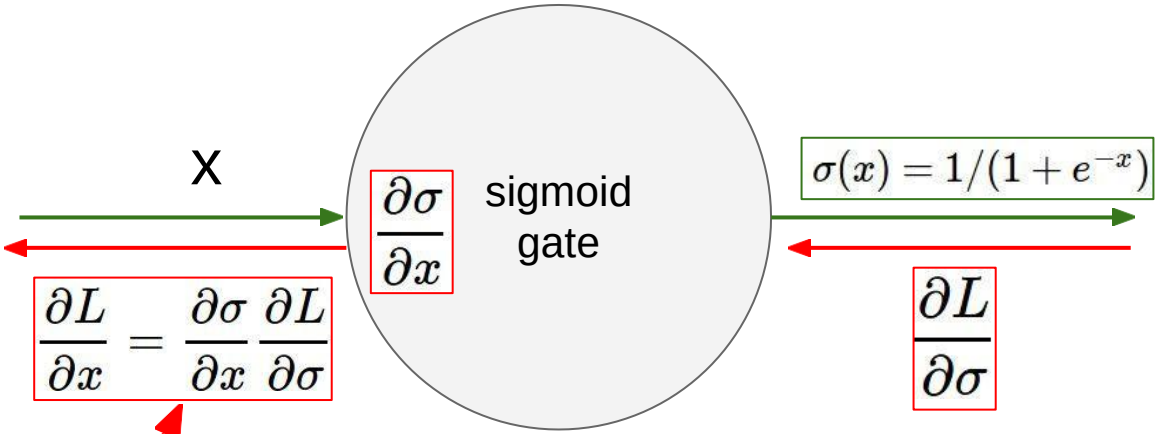
$$\frac{\partial \sigma(x)}{\partial x} = \sigma(x)\,(1-\sigma(x)) = 1(1-1) = 0$$

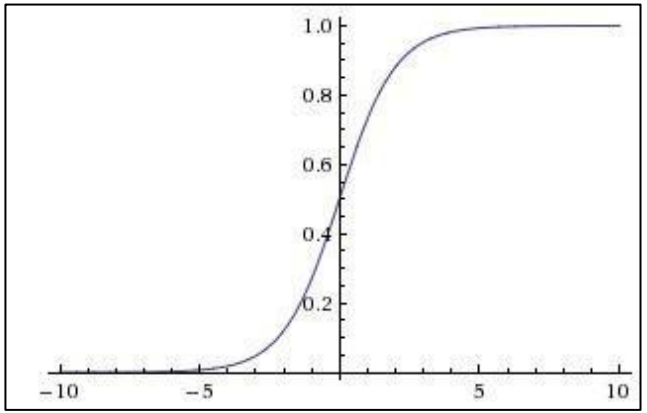




What h
What h
What h

$$\frac{\partial \sigma(x)}{\partial x} = \sigma(x)\,(1 - \sigma(x))$$





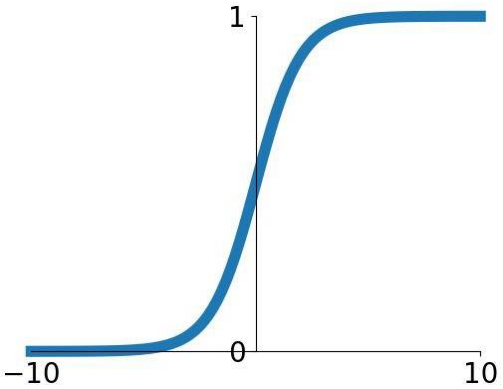
$$\frac{\partial \sigma(x)}{\partial x} = \sigma(x)\,(1 - \sigma(x))$$

Why is this a problem?

If all the gradients flowing back will be zero and weights will never change

Activation Functions

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



Sigmoid

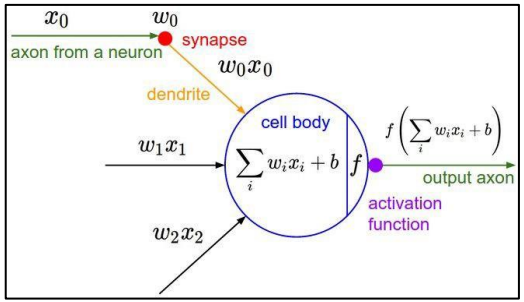
- Squashes numbers to range [0,1]
- Historically popular since they have nice interpretation as a saturating "firing rate" of a neuron

3 problems:

1. Saturated neurons "kill" the gradients
2. Sigmoid outputs are not zero-centered

Consider what happens when the input to a neuron is always positive...

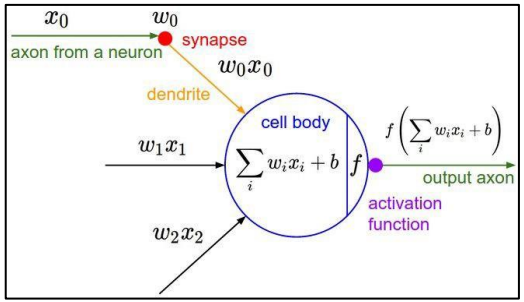
$$f\left(\sum_i w_i x_i + b\right)$$



What can we say about the gradients on **w**?

Consider what happens when the input to a neuron is always positive...

$$f\left(\sum_i w_i x_i + b\right)$$

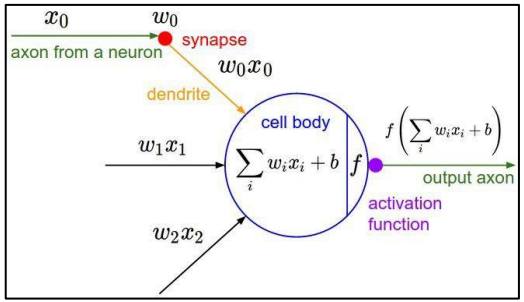


What can we say about the gradients on **w**?

$$\frac{\partial L}{\partial w} = \sigma(\textstyle\sum_i w_i x_i + b)(1 - \sigma(\textstyle\sum_i w_i x_i + b))x \times upstream\_gradient$$

Consider what happens when the input to a neuron is always positive...

$$f\left(\sum_i w_i x_i + b\right)$$



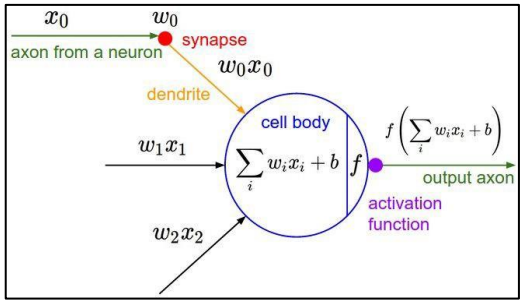
What can we say about the gradients on **w**?

We know that local gradient of sigmoid is always positive

$$\frac{\partial L}{\partial w} = \sigma\left(\sum_i w_i x_i + b\right)\left(1 - \sigma\left(\sum_i w_i x_i + b\right)\right) x \times upstream\_gradient$$

Consider what happens when the input to a neuron is always positive...

$$f\left(\sum_i w_i x_i + b\right)$$



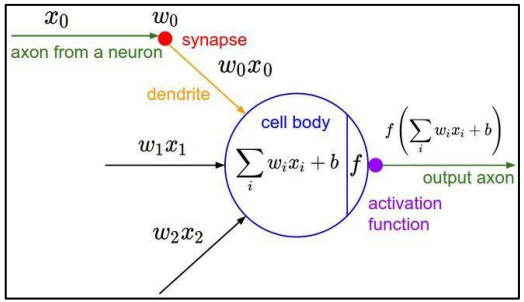
What can we say about the gradients on **w**?

We know that local gradient of sigmoid is always positive
We are assuming x is always positive

$$\frac{\partial L}{\partial w} = \sigma\left(\sum_i w_i x_i + b\right)\left(1 - \sigma\left(\sum_i w_i x_i + b\right)\right) x \times upstream\_gradient$$

Consider what happens when the input to a neuron is always positive...

$$f\left(\sum_i w_i x_i + b\right)$$



What can we say about the gradients on **w**?

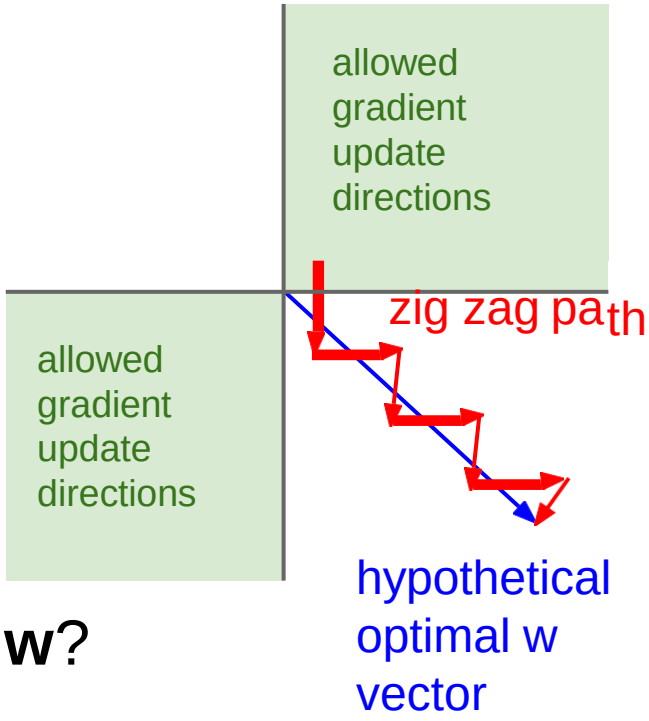
We know that local gradient of sigmoid is always positive
We are assuming x is always positive

So!! Sign of gradient **for all w_i** is the same as the sign of upstream scalar gradient!

$$\frac{\partial L}{\partial w} = \sigma\left(\sum_i w_i x_i + b\right)\left(1 - \sigma\left(\sum_i w_i x_i + b\right)\right)x \times upstream\_gradient$$

Consider what happens when the input to a neuron is always positive...

$$f\left(\sum_i w_i x_i + b\right)$$

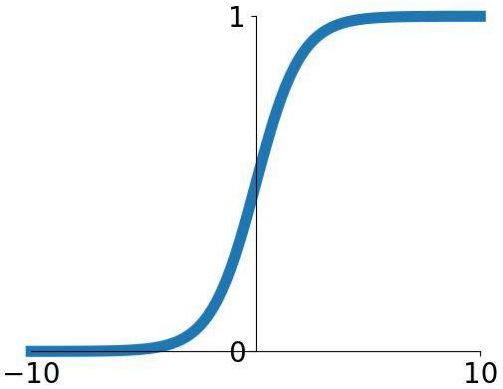


What can we say about the gradients on **w**?

Always all positive or all negative :(

Activation Functions

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



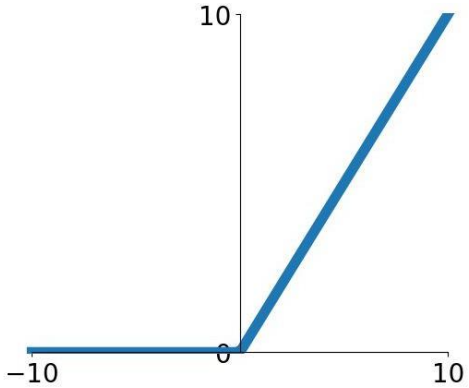
Sigmoid

- Squashes numbers to range [0,1]
- Historically popular since they have nice interpretation as a saturating “firing rate” of a neuron

3 problems:

1. Saturated neurons “kill” the gradients
2. Sigmoid outputs are not zero-centered
3. exp() is a bit compute expensive

Activation Functions



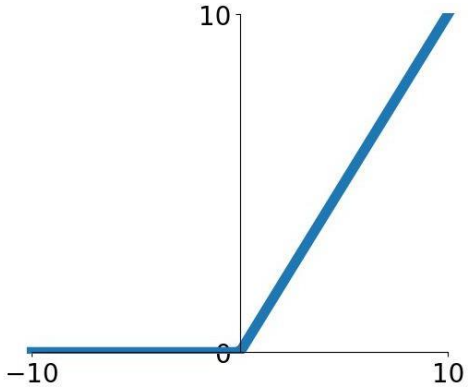
ReLU
(Rectified Linear Unit)

- Computes **f(x) = max(0,x)**

- Does not saturate (in +region)
- Very computationally efficient
- Converges much faster than sigmoid/tanh in practice (e.g. 6x)

[Krizhevsky et al., 2012]

Activation Functions



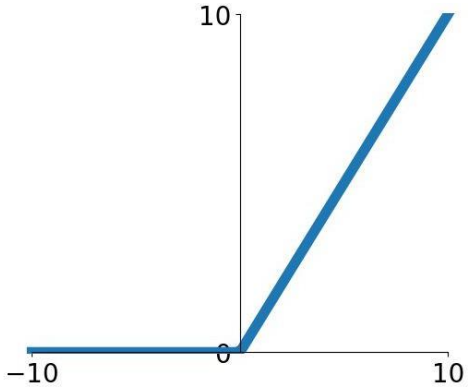
ReLU
(Rectified Linear Unit)

- Computes **f(x) = max(0,x)**

- Does not saturate (in +region)
- Very computationally efficient
- Converges much faster than sigmoid/tanh in practice (e.g. 6x)

- Not zero-centered output

Activation Functions



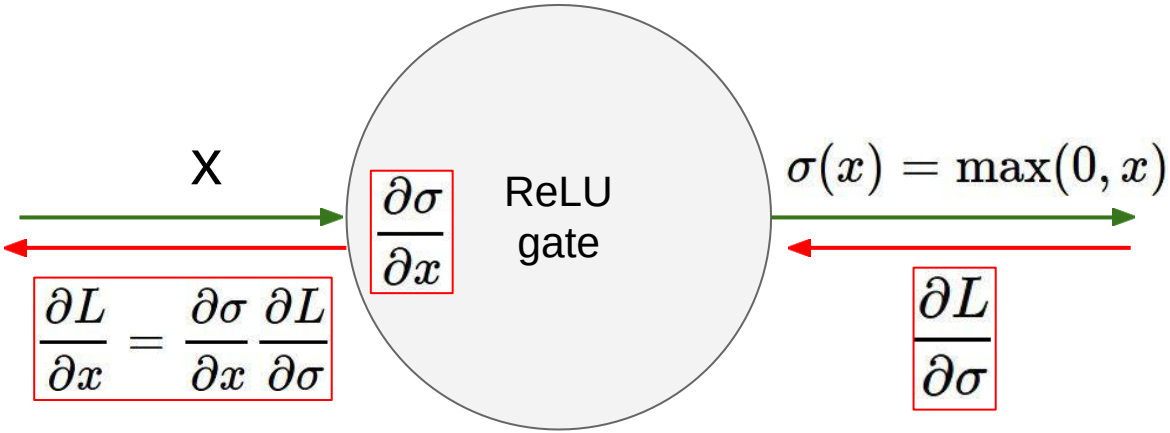
ReLU
(Rectified Linear Unit)

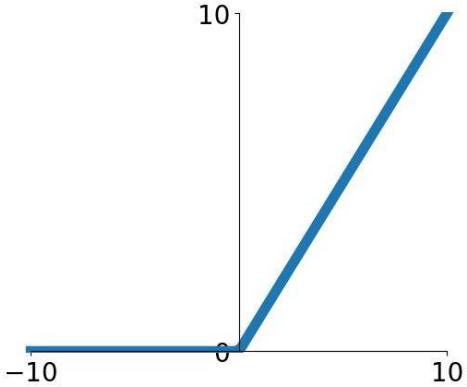
- Computes **f(x) = max(0,x)**

- Does not saturate (in +region)
- Very computationally efficient
- Converges much faster than sigmoid/tanh in practice (e.g. 6x)

- Not zero-centered output
- An annoyance:

hint: what is the gradient when x < 0?



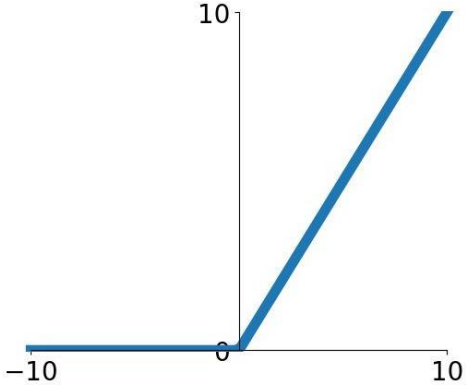


What happens when x = -10?
What happens when x = 0?
What happens when x = 10?

If the input is always negative (bad learning rate, inappropriate data preprocessing) – they are “died” .

Why does ReLU can still learn without the negative response?

Sparse Activation: ReLU outputs zero for any negative input, leading to a portion of neurons in the network not activating. This sparsity in activation patterns improves computational efficiency and regularization.

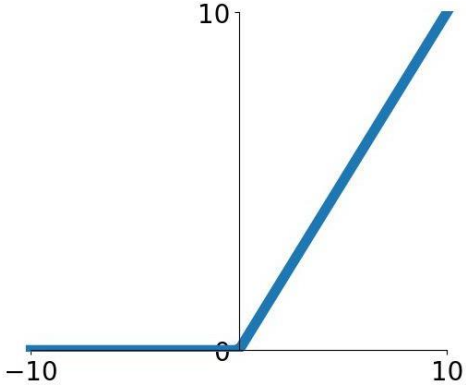


Non-saturation: The ReLU function is linear for positive inputs, meaning it does not saturate (or flatline) as the input gets large, unlike the sigmoid or tanh functions.

.

Why does ReLU can still learn without the negative response?

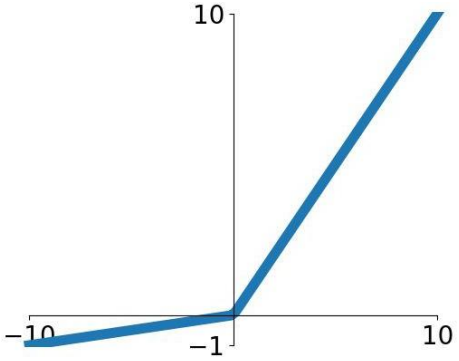
Enhancement of Positive Features:
Although ReLU does not directly process negative inputs (setting them to zero), it can emphasize positive features through the adjustment of weights learned by the network.



Collaboration in Multilayer Networks: even though individual ReLU units cannot process negative inputs, multiple layers of ReLU units can interact through their weights and structure to abstract useful information and features from complex inputs.

Activation Functions

[Mass et al., 2013]
[He et al., 2015]



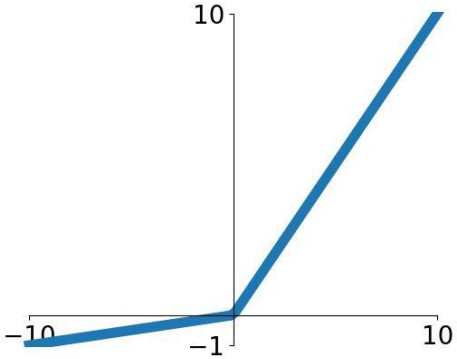
- Does not saturate
- Computationally efficient
- Converges much faster than sigmoid/tanh in practice! (e.g. 6x)
- **will not “die”.**

Leaky ReLU

$$f(x) = \max(0.01x, x)$$

Activation Functions

[Mass et al., 2013]
[He et al., 2015]



- Does not saturate
- Computationally efficient
- Converges much faster than sigmoid/tanh in practice! (e.g. 6x)
- **will not "die".**

Leaky ReLU

$$f(x) = \max(0.01x, x)$$

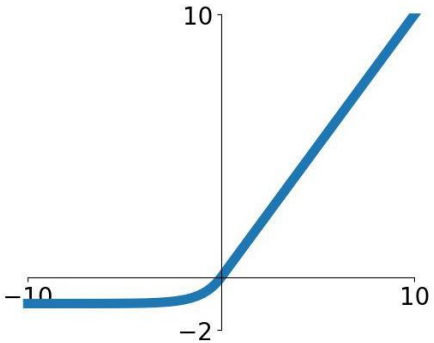
Parametric Rectifier (PReLU)

$$f(x) = \max(\alpha x, x)$$

backprop into α (parameter)

[Clevert et al., 2015]

Exponential Linear Units (ELU)



- All benefits of ReLU
- Closer to zero mean outputs
- Negative saturation regime compared with Leaky ReLU adds some robustness to noise

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha\,(\exp(x) - 1) & \text{if } x \leq 0 \end{cases}$$

(Alpha default = 1)

- Computation requires exp()

TLDR: In practice:

- Use ReLU. Be careful with your learning rates
- Try out Leaky ReLU / Maxout / ELU
  - To squeeze out some marginal gains
- Don't use sigmoid or tanh

We will cover more about neural network training once we finish CNN, RNN, and GNN

Important Activation Interview Questions

Question: What are the limitations of backpropagation?

Important Activation Interview Questions

Question: What are the limitations of backpropagation?

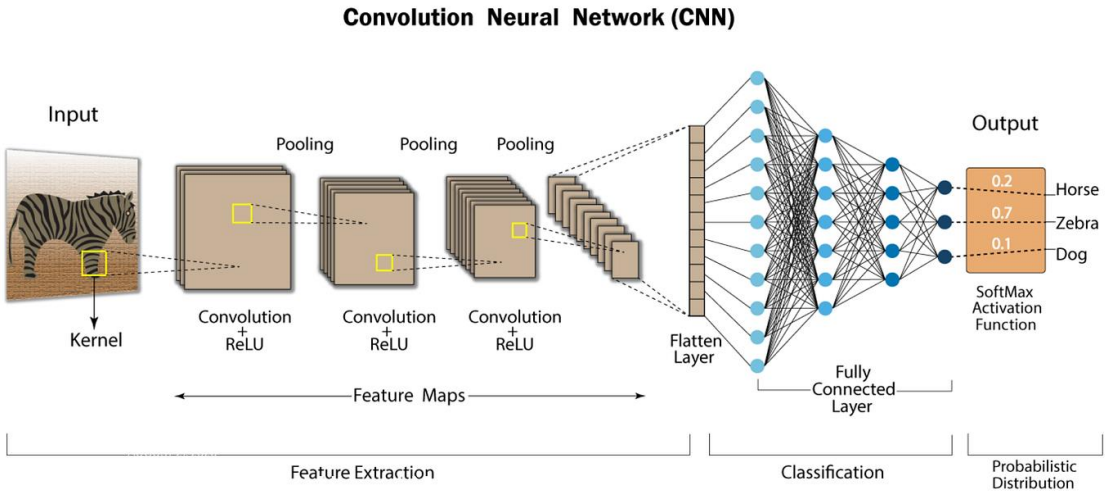
Answer:

- **Vanishing Gradient Problem**: In deep networks, gradients can become very small, effectively preventing the weights from being updated during training.
- **Exploding Gradient Problem**: Conversely, gradients can become excessively large, leading to divergent behavior.
- **Dependency on Data and Initial Weights**: The efficiency of backpropagation heavily relies on the quality of the data and the initial weight settings.
- **Local Minima**: Backpropagation can lead the network to converge to local minima rather than the global minimum.
- **Computational Intensity**: For large networks, backpropagation can be computationally expensive.

Next Blocks – From Simple NN to CNN, RNN, GNN

Real-world data is more complex than simple features:

- For a 1024x1080 picture – how many features does it have?
- For stock data with time dependency, how to consider the time dependency?
- For social networks with strong connectivity, how to capture it?

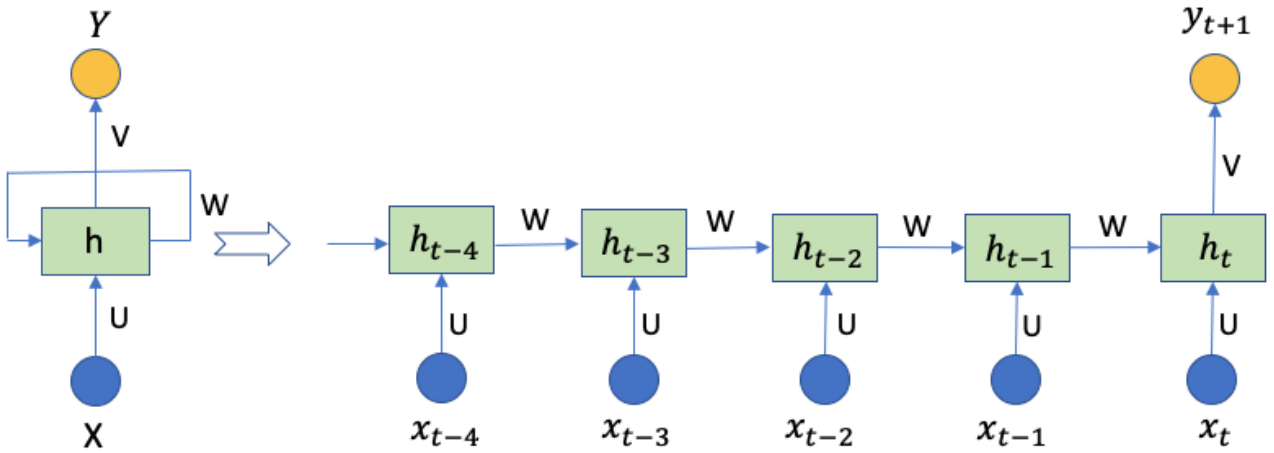


Convolution Neural Networks (CNN) for images and videos

Next Blocks – From Simple NN to CNN, RNN, GNN

Real-world data is more complex than simple features:

- For a 1024x1080 picture – how many features does it have?
- For stock data with time dependency, how to consider the time dependency?
- For social networks with strong connectivity, how to capture it?

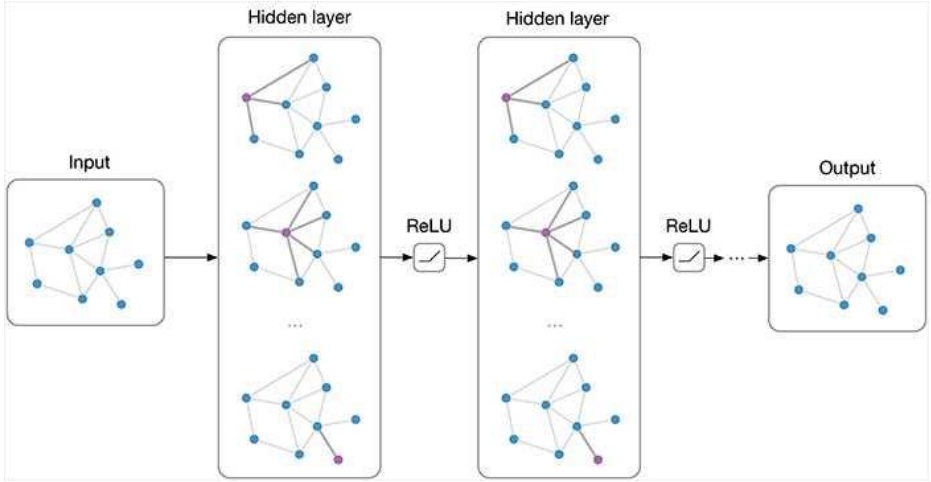


Recurrent neural networks (RNN) for sequences

Next Blocks – From Simple NN to CNN, RNN, GNN

Real-world data is more complex than simple features:

- For a 1024x1080 picture – how many features does it have?
- For stock data with time dependency, how to consider the time dependency?
- For social networks with strong connectivity, how to capture it?



Graph neural networks (GNN) for graphs