Lecture 7: Neural Networks (Part 1)

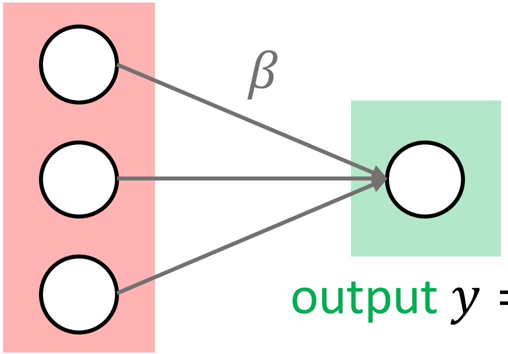
CIS 4190/5190

Spring 2025

Slides adapted from Chris Callison-Berch and Luke Zettlemoyer and Fei-Fei Li

So far in this class

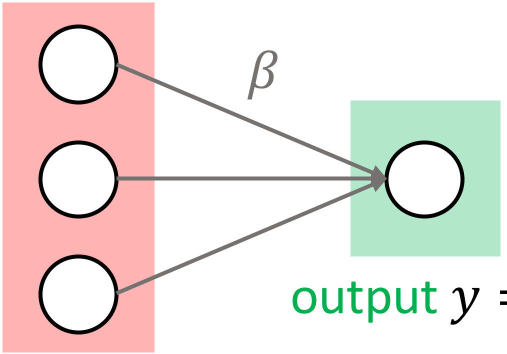
Linear Regression



output $y = \beta^\top x$

input x

Binary Classification

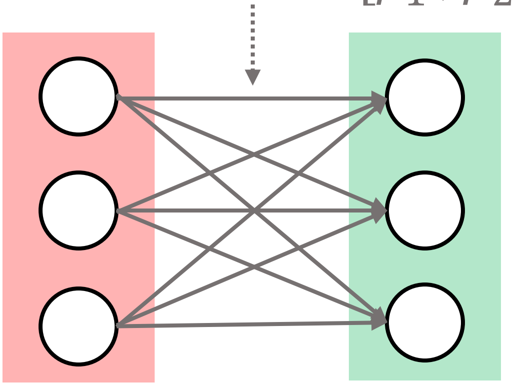


output $y = \sigma(\beta^\top x)$

input x

Multi-Class Classification

$W = [\beta_1^\top; \beta_2^\top; \ldots; \beta_C^\top]$



input x

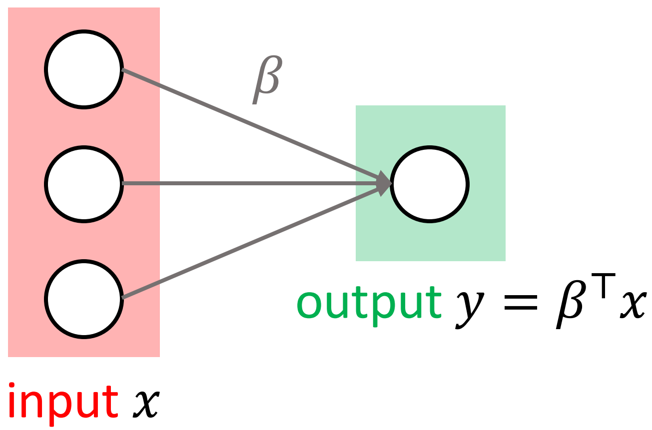
output $y = softmax(Wx)$

$$p_W(y = c \mid x) = \frac{e^{\beta_c^\top x}}{\sum_{y'} e^{\beta_{y'}^\top x}}$$

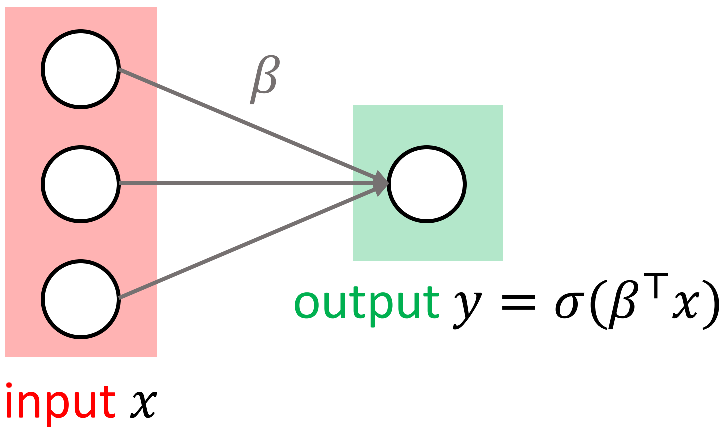
A Unifying View

Linear transformation of input features followed by an activation function

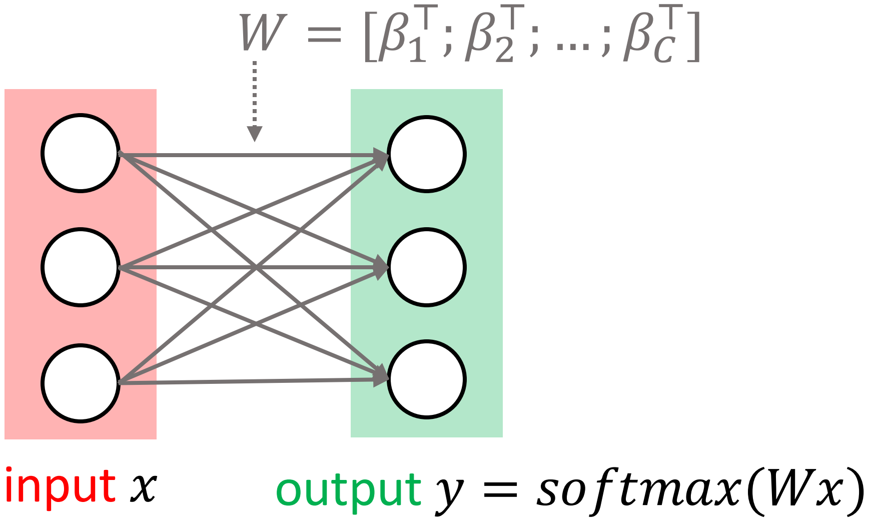
Linear Regression



Binary Classification



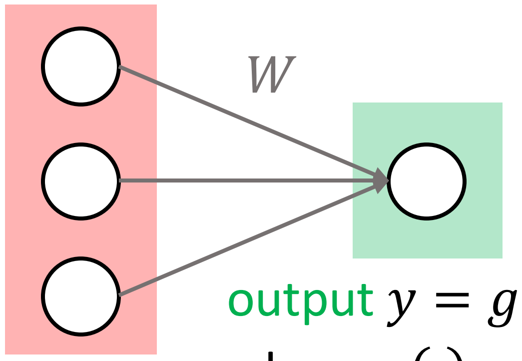
Multi-Class Classification



A Unifying View

Linear transformation of input features (Wx) followed by an activation function $g(\cdot)$

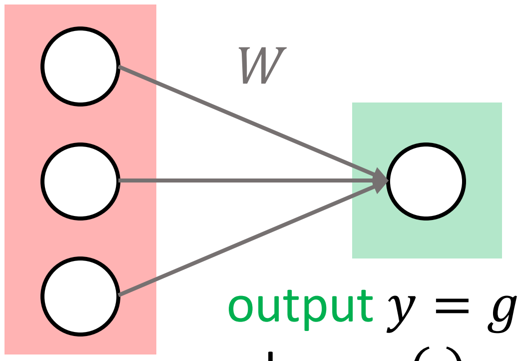
Linear Regression



input x

output $y = g(Wx)$
where $g(\cdot) = \cdot$

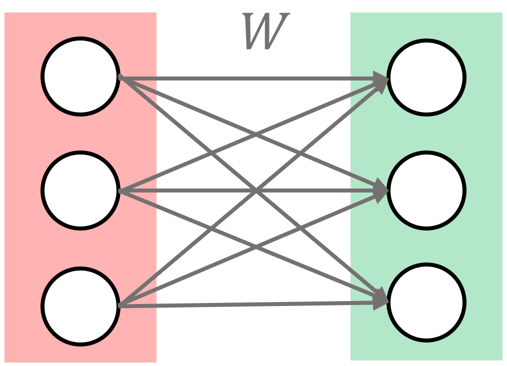
Binary Classification



input x

output $y = g(Wx)$
where $g(\cdot) = \sigma(\cdot)$

Multi-Class Classification

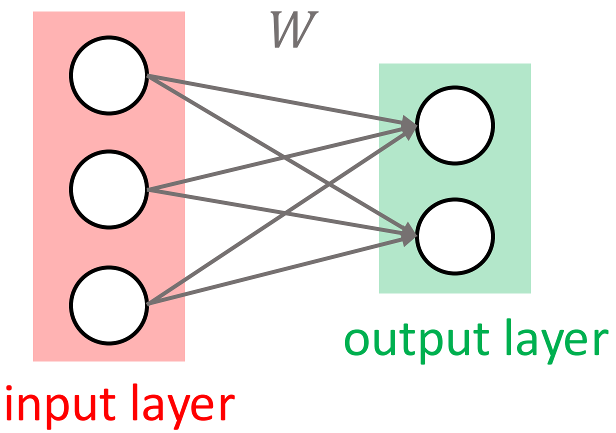


input x

output $y = g(Wx)$
where $g(\cdot) = softmax(\cdot)$

$W \in \mathbb{R}^{C \times D}$ where D is the input dimension and C is the output dimension.

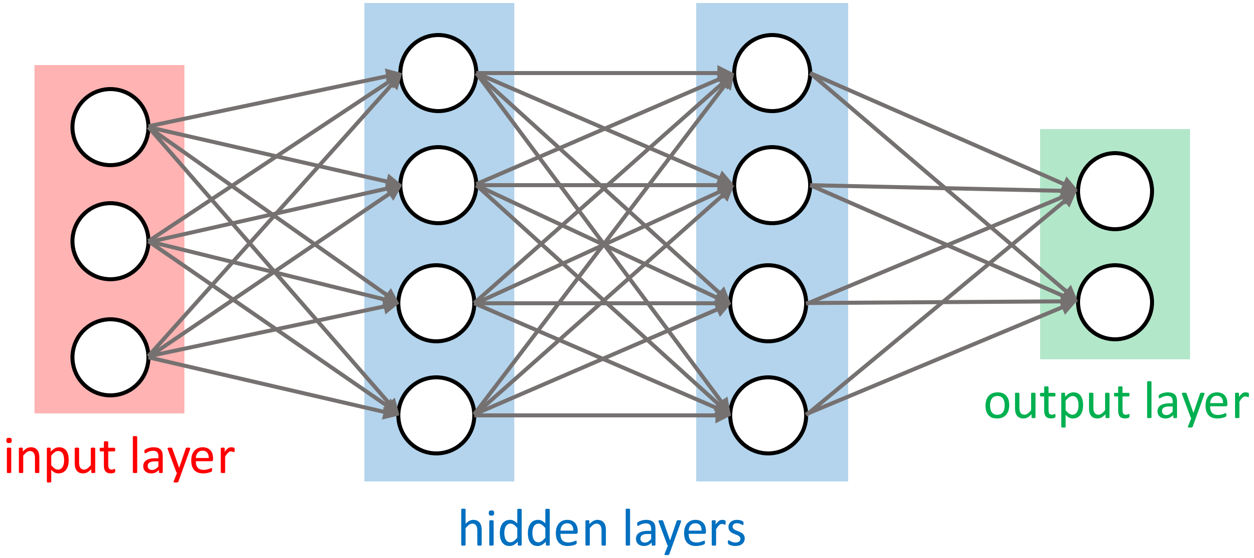
A Unifying View: Single-Layer Neural Network

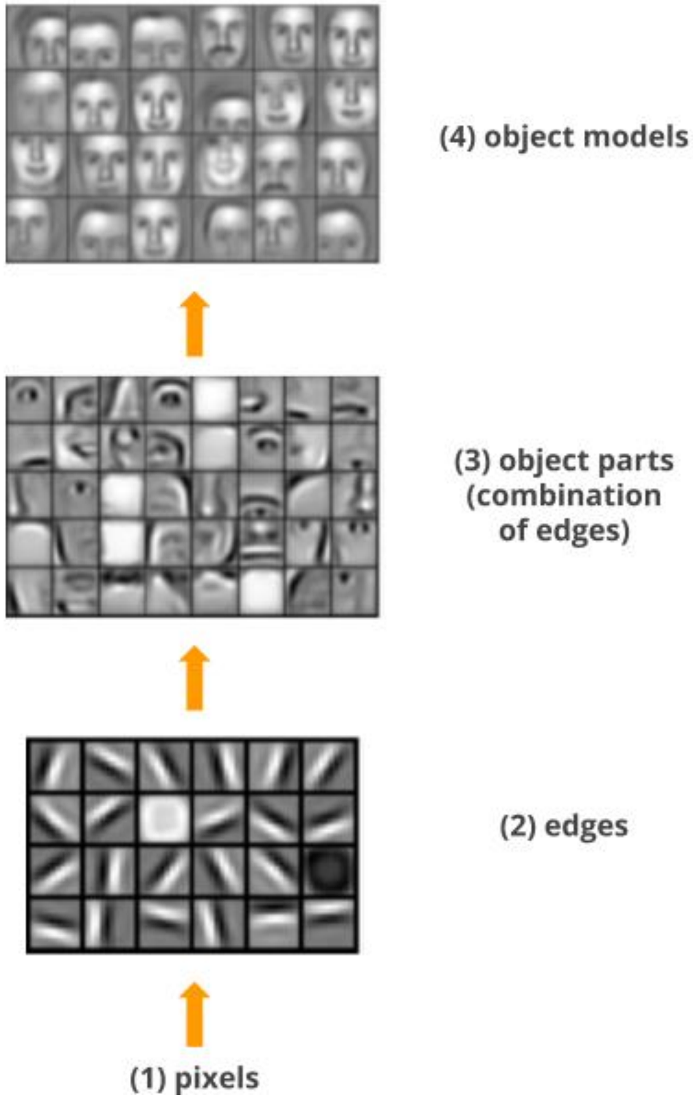


- Challenge: it needs "good" input features
- Most ML work before focused on hand designing features

The “Promise” of Deep Neural Networks

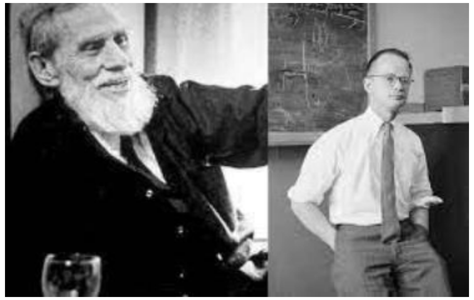
- **Representation Learning**: automatically learn good features for tasks
- **Deep Learning**: learn multiple levels of representation at increasing levels of complexity

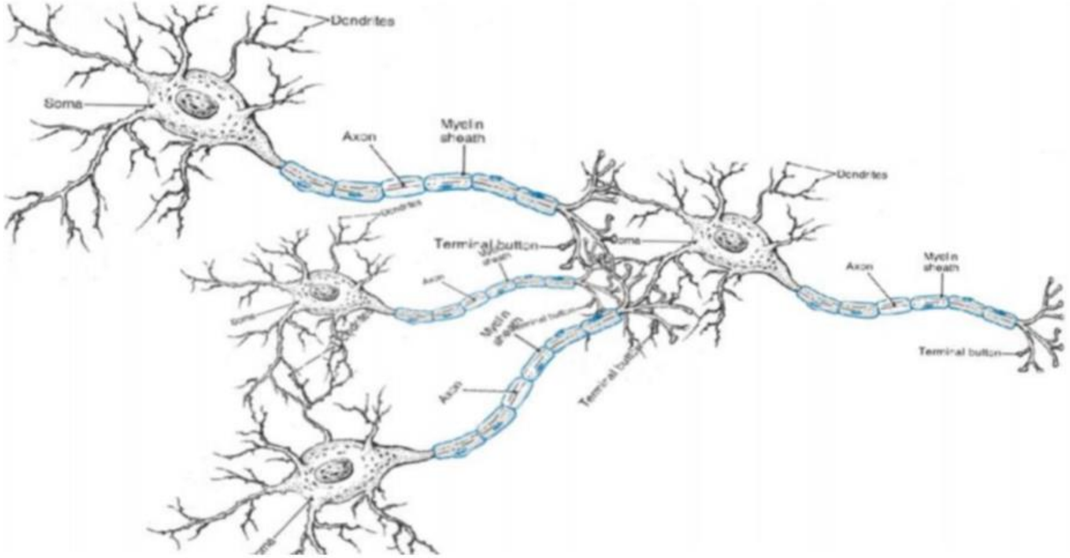




Inspired by Simplified Models of Brain Neurons

- **1943:** Perceptron model (McCulloch & Pitts)
  - Intended as theoretical model of biological neurons





"Dark Ages"

- **1969:** Perceptrons cannot learn XOR (Minsky & Papert)
  - Highly controversial (may have helped cause "AI winter")

- **1998:** Convolutional neural networks for MNIST (Lecun)
  - Human-level performance on handwritten digit recognition

- **1997**: Long Short-term Memory Networks (Hochreiter and Schmidhuber)

Extremely
Similar Today

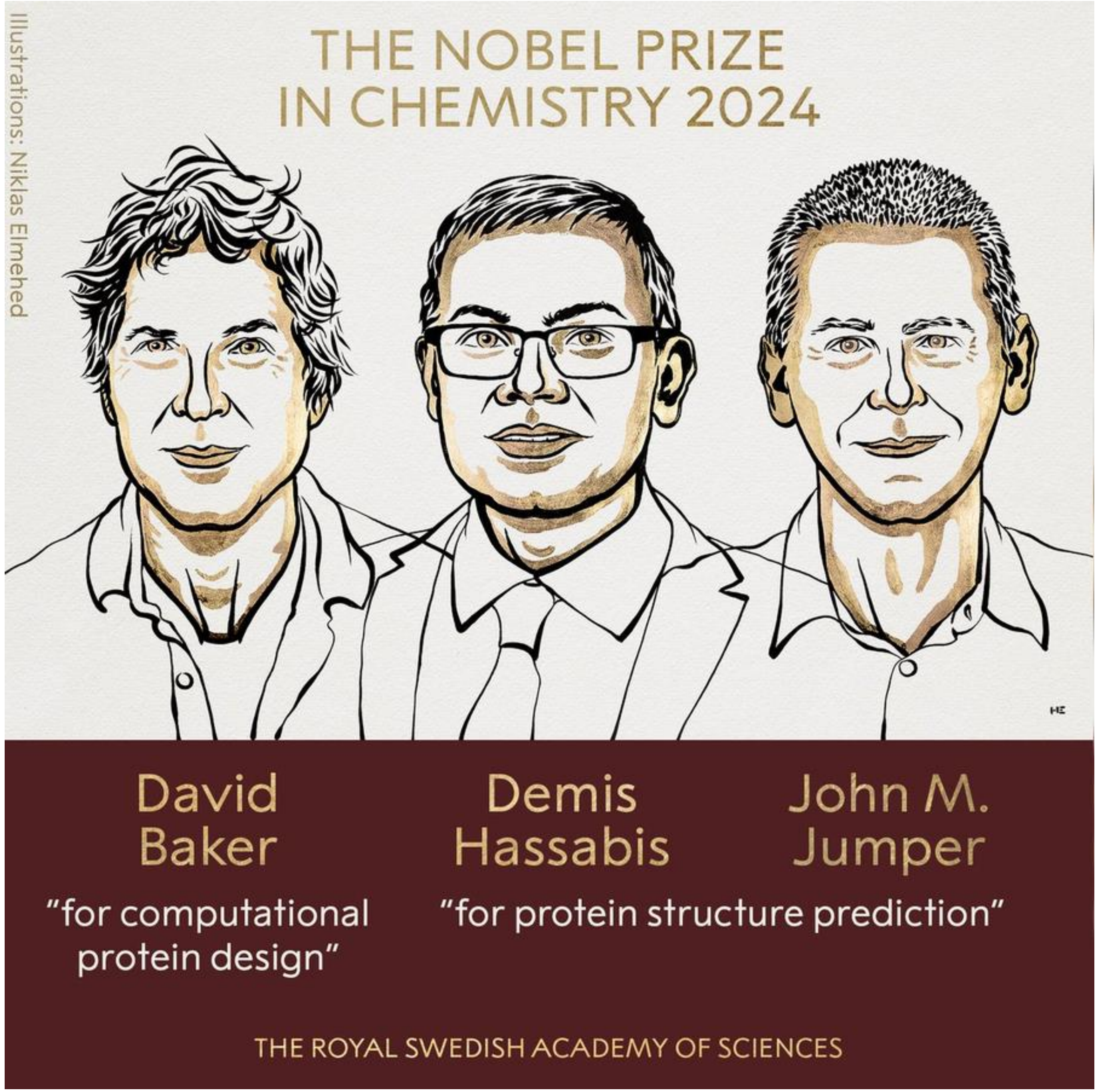
2012 - NOW

- **2012:** ImageNet breakthrough (Krizhevsky, Sutskever, & Hinton)
  - Reduced error on image classification by 50%

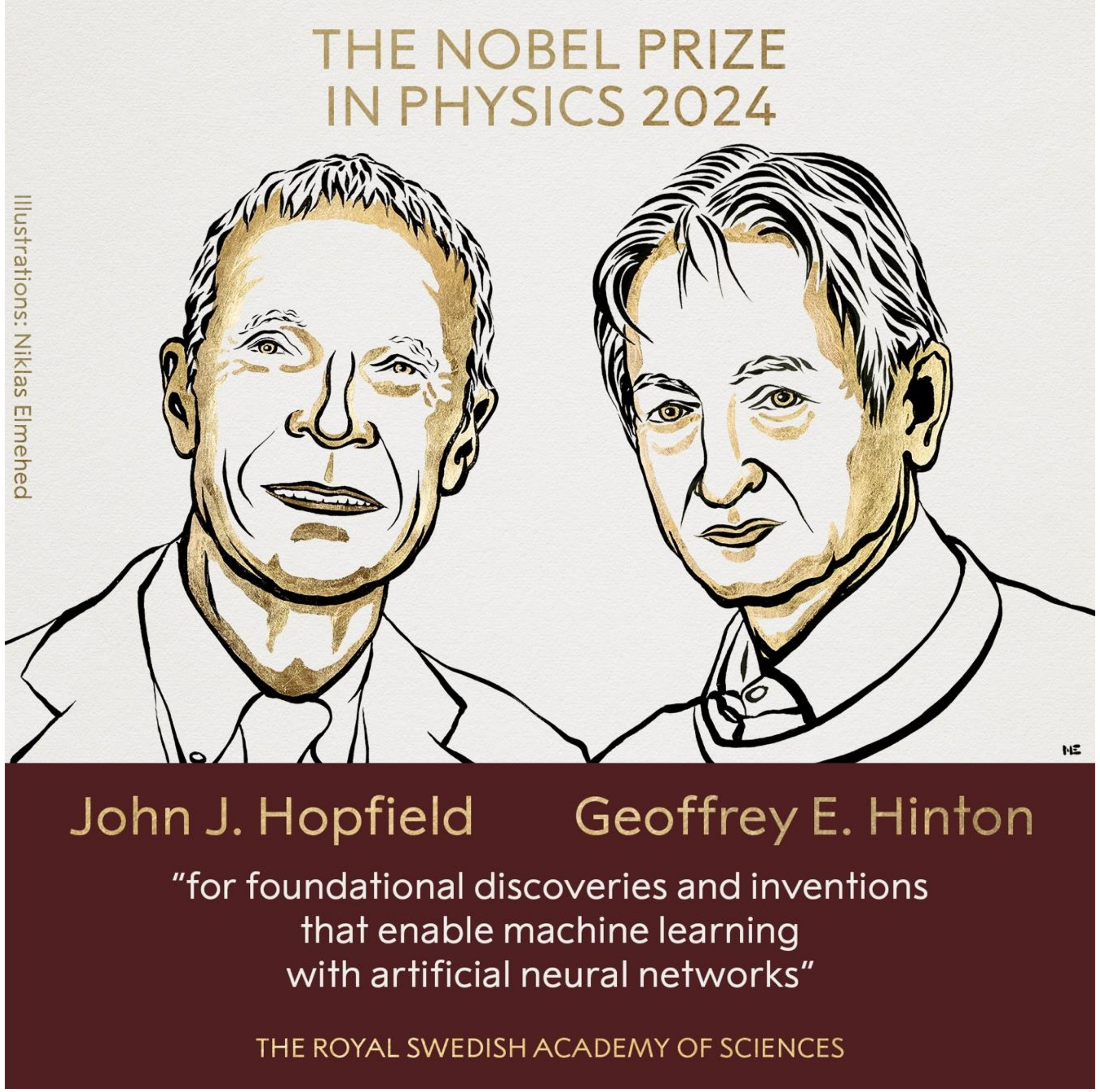
- **2017:** Transformer architecture (Vaswani et al.)

- **2018:** Turing award (Bengio, Hinton, & Lecun)

The Nobel Prize in Chemistry 2024



The Nobel Prize in Physics 2024



2012 - NOW

- **2012:** ImageNet breakthrough (Krizhevsky, Sutskever, & Hinton)
  - Reduced error on image classification by 50%
- **2017:** Transformer architecture (Vaswani et al.)
- **2018:** Turing award (Bengio, Hinton, & Lecun)
- **2024:** The Nobel Prize in Physics & Chemistry
- **Generative AI & LLMs… To be continued?**

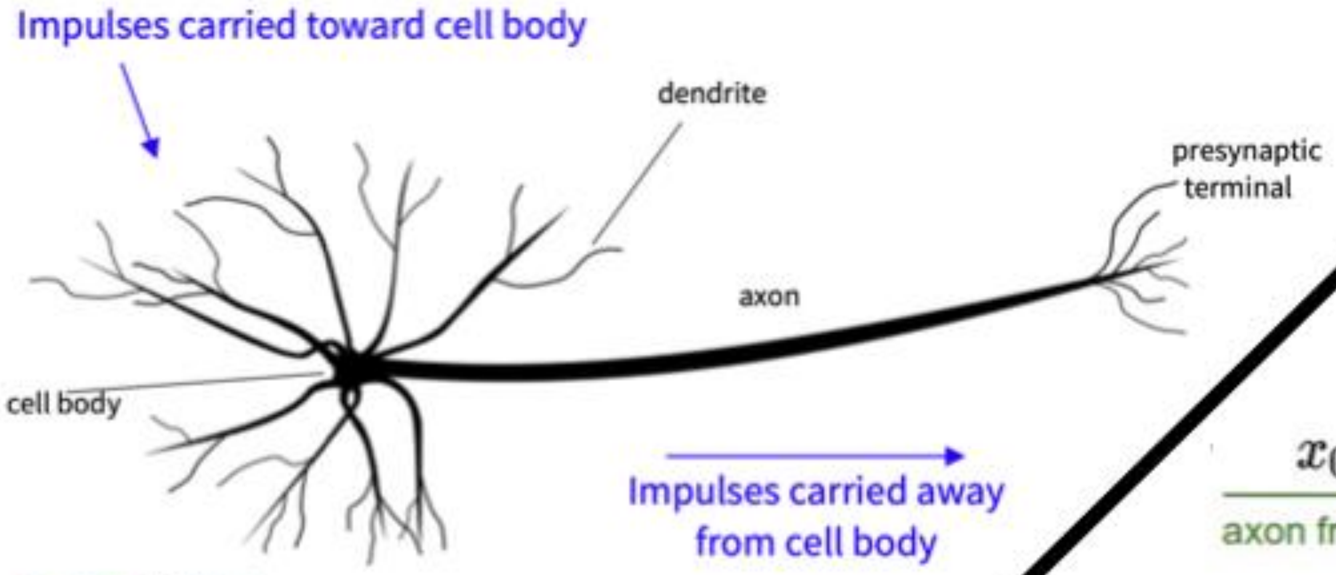
Why Wasn't it Working Before?

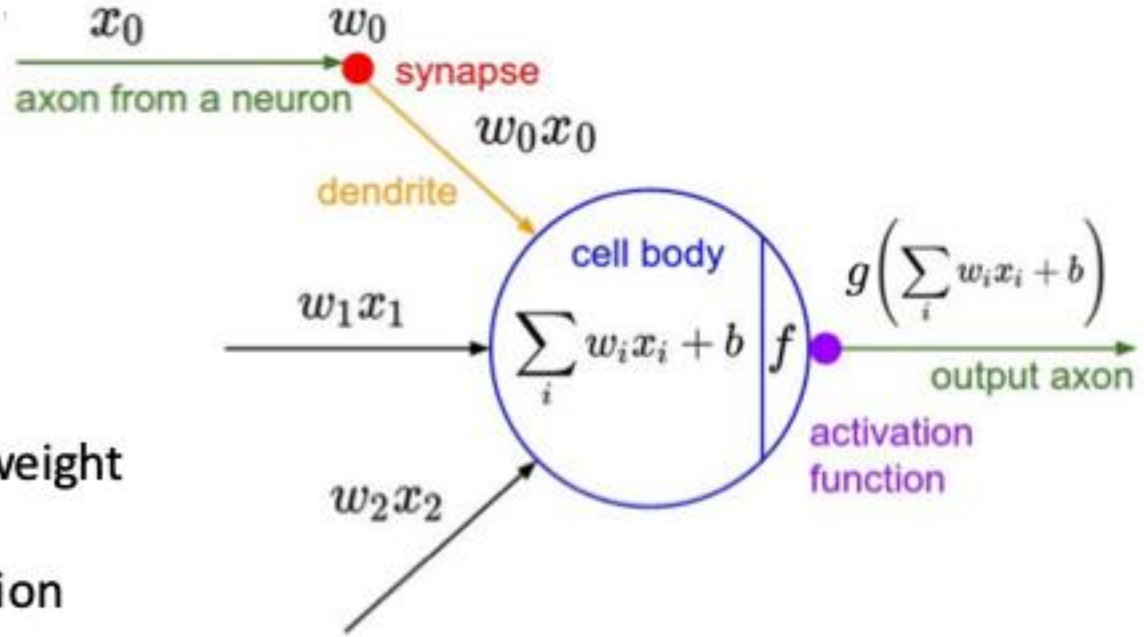
- Small Datasets & Less Capable Hardware
  - Machine translation needs millions of sentences to see improvements
- Missing bag of tricks for optimization ← **Next lecture**
  - Regularization like Dropout
- Some domain specific tricks & architectures ← **Topics for 2nd half of the semester**
  - Word embedding for NLP

Today’s Lecture

- Model
  - Feedforward Neural Networks
- Loss functions
- Optimization
  - Stochastic Gradient Descent
  - Back-Propagation for Computing Gradients

Simplified Brain Modeling

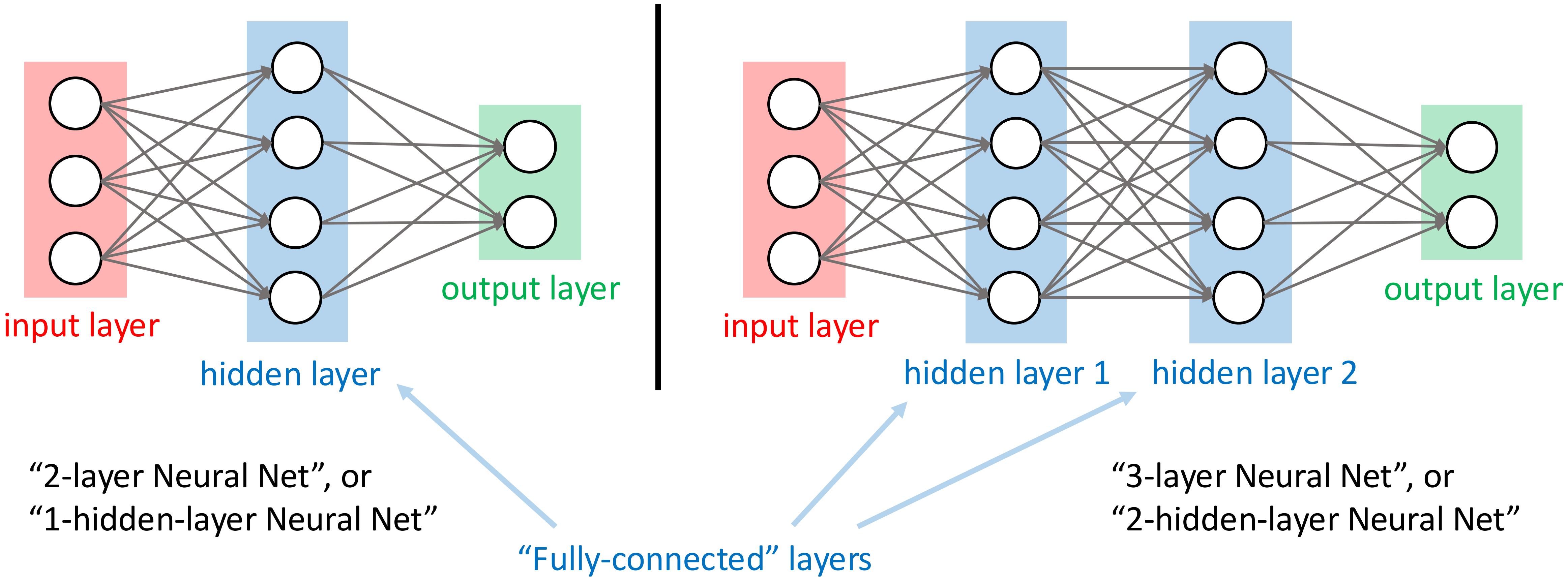




1. Takes scalar inputs
2. Multiply each input by a weight
3. Sum all weighted input
4. Apply a non-linear activation

Feed-Forward Neural Networks

- Signals move in one direction – forward – with no cycles or loops.
- Also called Multi-Layer Perceptrons (MLP)



Matrix Notation

1-layer Neural Net: $y = W_1 x$

2-layer Neural Net: $y = W_2 g(W_1 x)$

3-layer Neural Net: $y = W_3 g(W_2 g(W_1 x))$

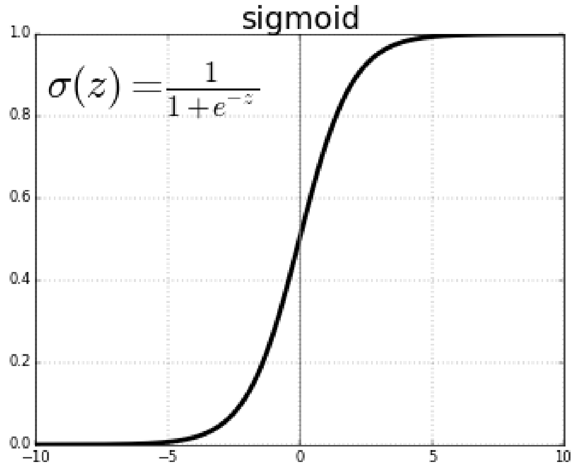
$x \in \mathbb{R}^D, W_1 \in \mathbb{R}^{H_1 \times D}, W_2 \in \mathbb{R}^{H_2 \times H_1}, W_3 \in \mathbb{R}^{H_3 \times H_2}$

g is a non-linear activation function for hidden layers

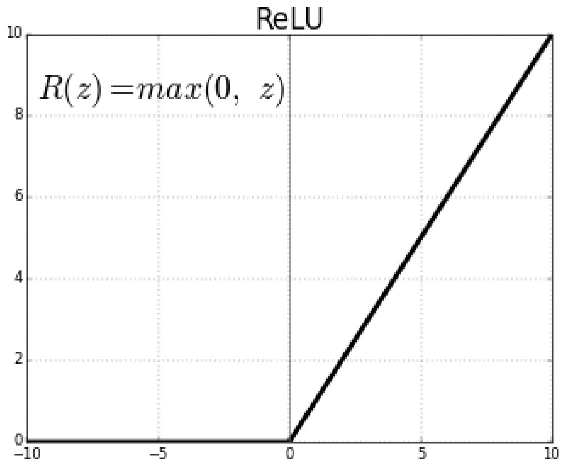
(In practice we will usually add a learnable bias at each layer as well)

Non-Linearity g

- Sigmoid activation function:
  - Outputs values between 0 and 1
  - Probability of neuron firing/activated



- ReLU (Rectified Linear Unit):
  - Efficient computation
  - Doesn't saturate
  - Most commonly used today



Why Non-Linearity?

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

2-layer Neural Net: $y = W_2 g(W_1 x)$ → $y = W_2 W_1 x$

3-layer Neural Net: $y = W_3 g(W_2 g(W_1 x))$ → $y = W_3 W_2 W_1 x$

A: We would end up with linear classifier!

Non-Linearities are important for learning features/representations with <u>increasing levels of complexity</u>

Model Capacity

- Capacity of a feed-forward neural network is affected by both:
  - Depth: number of hidden layers
  - Width: number of neurons in each hidden layer
- More neurons = more capacity

Today's Agenda

- Model
  - Feedforward Neural Networks
- Loss functions
- Optimization
  - Gradient Descent
  - Back-Propagation for Computing Gradients

Today's Lecture

- Model
  - Feedforward Neural Networks
- Loss functions
- Optimization
  - Stochastic Gradient Descent
  - Back-Propagation for Computing Gradients

Loss Functions

- Same as single-layer models (i.e., linear and logistic regression)
- Regression:
  - MSE loss: $\mathcal{L}_{\mathrm{MSE}}(y, y^*) = (y - y^*)^2$
- Classification:
  - Binary cross entropy for binary classification:

$$\mathcal{L}(y, y^*) = -y^* \log y - (1 - y^*) \log(1 - y)$$

  - Cross entropy for multi-class classification:

$$\mathcal{L}(y, y^*) = -\sum_{i=1}^{C} y_i^* \log y_i$$

Today's Agenda

- Model
  - Feedforward Neural Networks
- Loss functions
- Optimization
  - Stochastic Gradient Descent
  - Back-Propagation for Computing Gradients

Optimization

Solve for $\theta^* = \underset{\theta}{\operatorname{argmin}}\, L(\hat{y}, y)$

Q: Don't I have to optimize differently for different $L(\cdot)$?

A: No, just use gradient descent. It is the most general optimization approach we know.

Q: But what if $L(\cdot)$ is non-convex in W?

A: It almost surely is. Do gradient descent anyway. Just make sure everything is differentiable.

Computing Gradients

- You could write down the full function and calculate the gradients for all the weights manually.
  - It takes a lot of time and paper
  - Change loss function (e.g., add L2/L1) → Need to compute from scratch again

- Better Idea:
  - Use computational graph and chain rule of gradient calculation

Backpropagation

- It's taking derivatives and applying chain rule! $\frac{\partial F(G(x))}{\partial x} = \frac{\partial F(G(x))}{\partial G} * \frac{\partial G(x)}{\partial x}$

- We will re-use derivatives computed for higher layers in computing derivates for lower layers so as to minimize computation

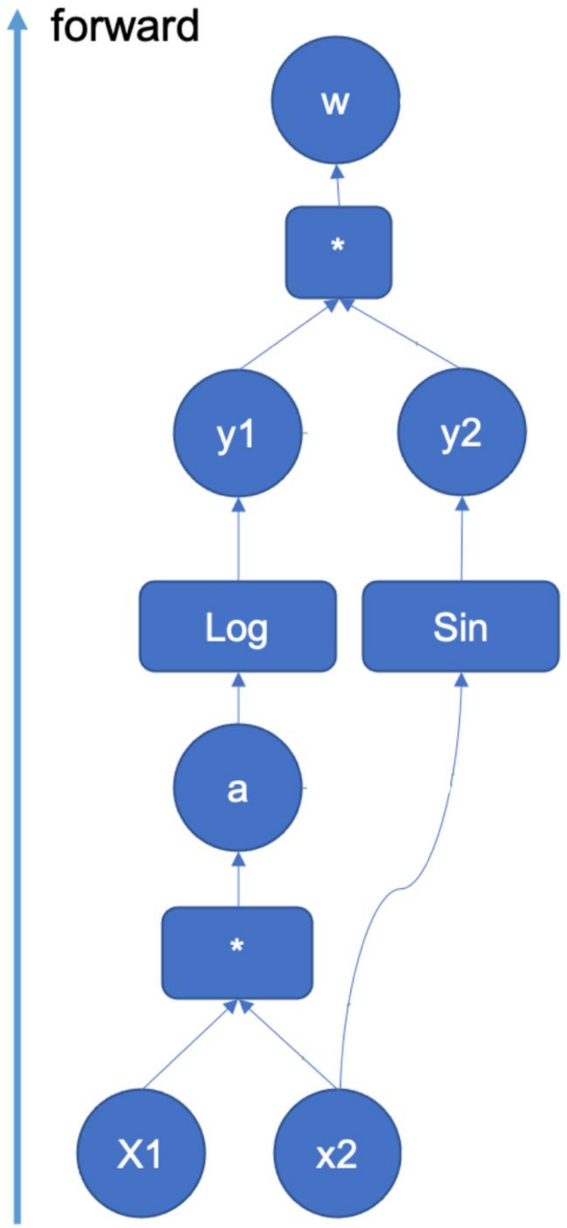
- Good news is that modern automatic differentiation tools did all for you!
  - Implementing backprop by hand is like programming in assembly languages.

Computational Graph

- Break down function computation:
  - Data (input, output, and intermediate)
  - Operators (e.g., addition, multiplication)

- Consider the following function:

$$w = \log(x_1 x_2) \sin(x_2)$$



https://pytorch.org/blog/computational-graphs-constructed-in-pytorch/

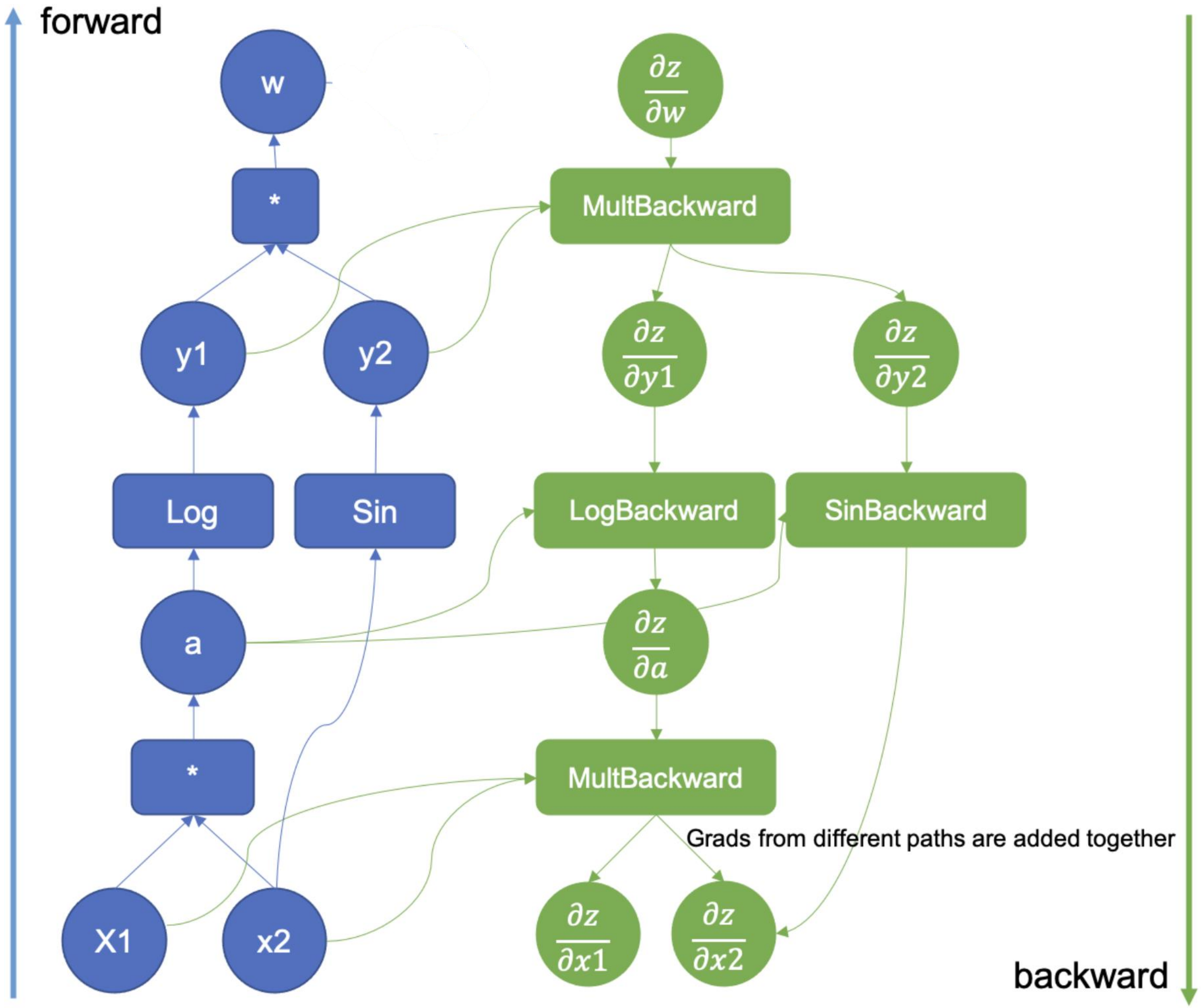
Full Graph

Requirement for each operator:

Forward: compute their output function given input.

Backward: compute the gradient of their output w.r.t. each input.

If all operators can do forward & backward computation, we can compute the derivative of the output with respect to any input, <u>procedurally</u>.

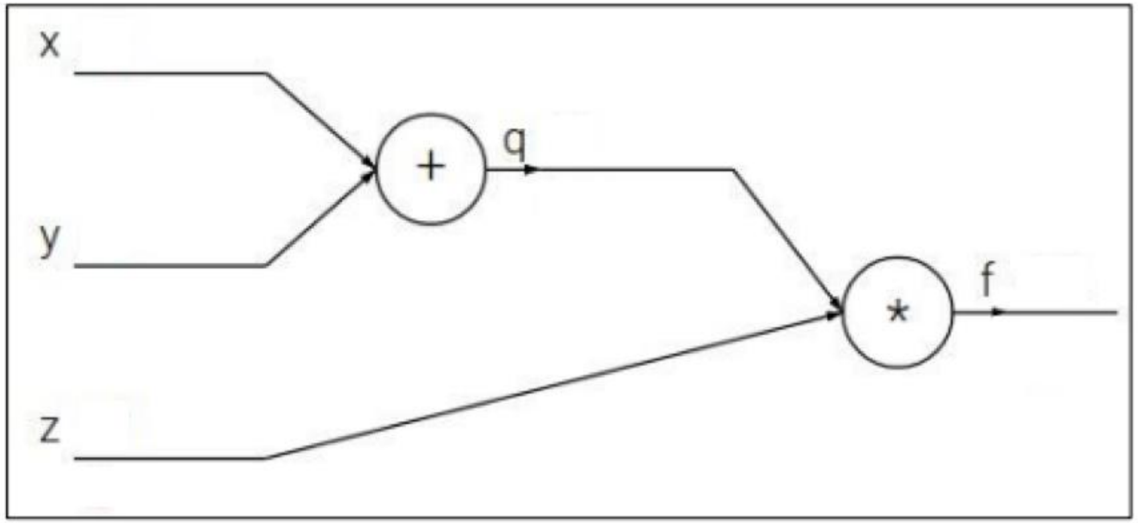


https://pytorch.org/blog/computational-graphs-constructed-in-pytorch/

Example On Simple Function

gradients to maximize f

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



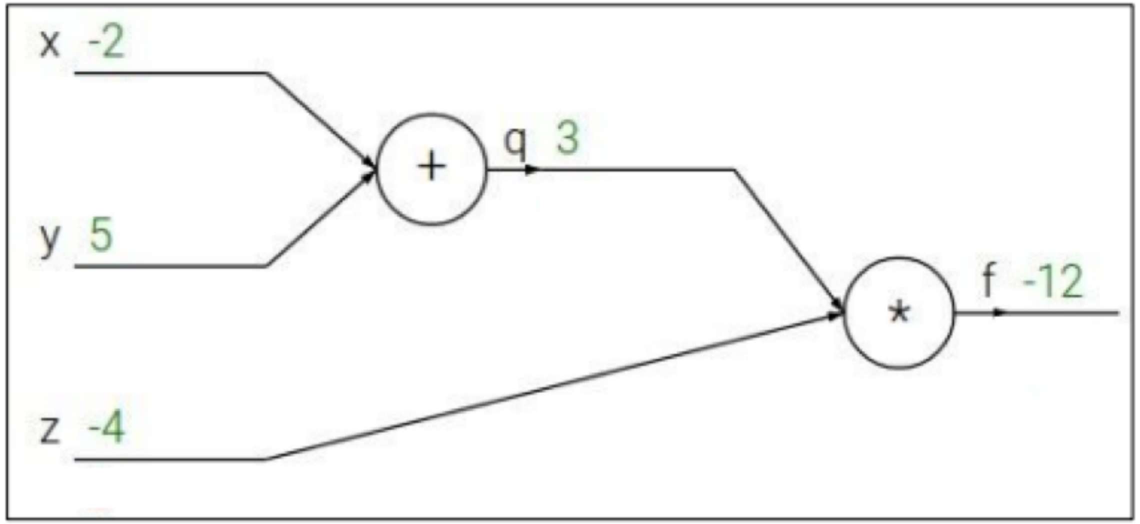
(circles represents operators here)

Example On Simple Function

gradients to maximize f (at a point)

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

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



Example On Simple Function

gradients to maximize f (at a point)

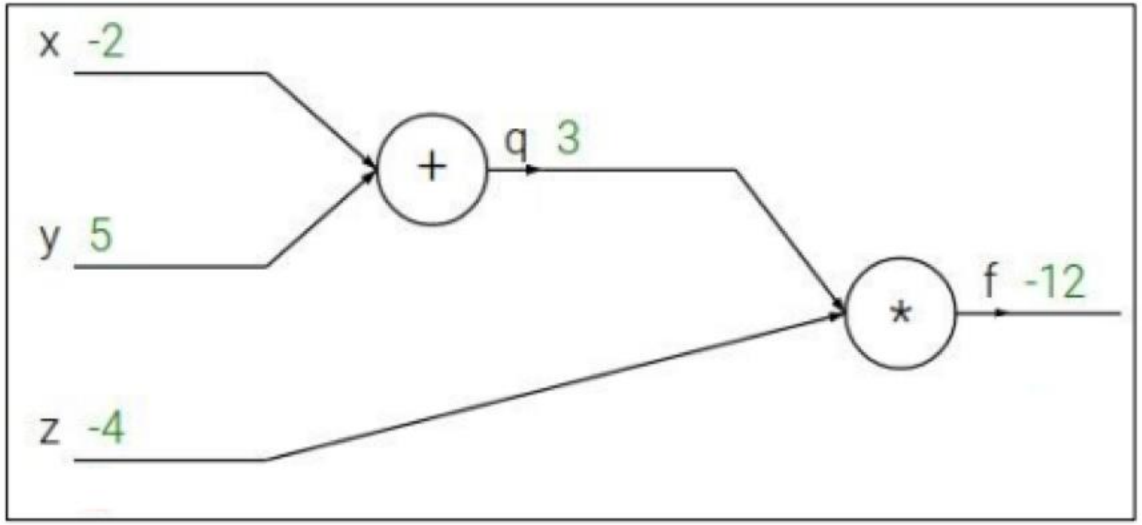
$f(x, y, z) = (x + y)z$

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

Node Gradients

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

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



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

Example On Simple Function

gradients to maximize f (at a point)

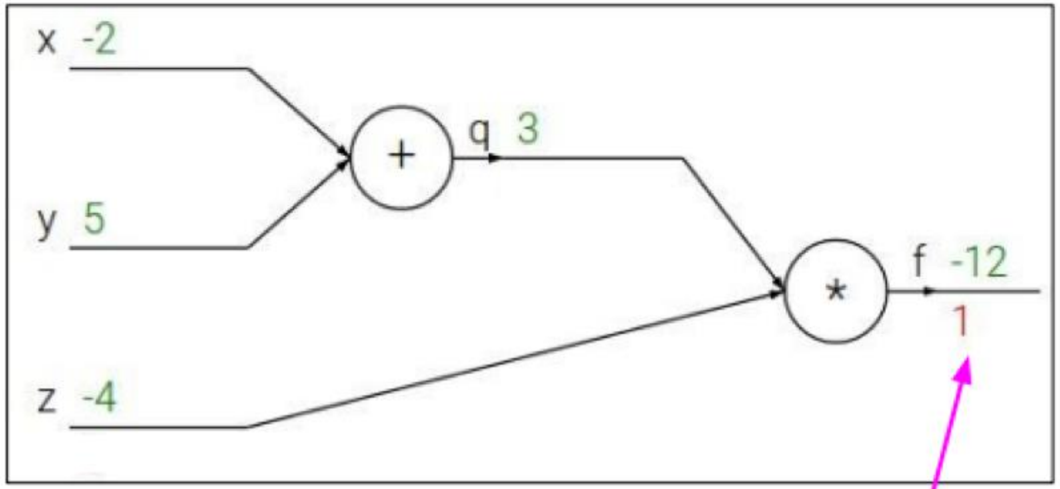
$$f(x,y,z) = (x+y)z$$

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

Node Gradients

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

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



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

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

Example On Simple Function

gradients to maximize f (at a point)

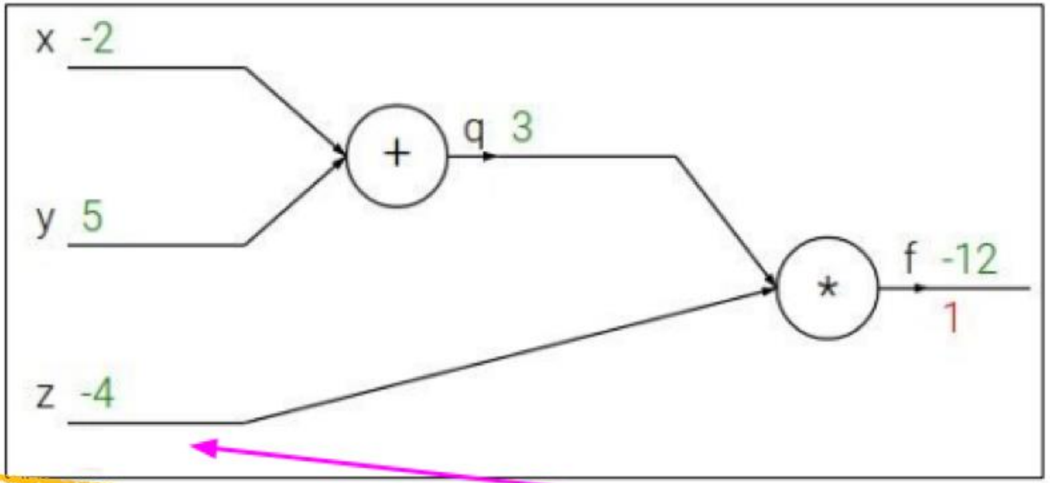
$$f(x, y, z) = (x + y)z$$

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

Node Gradients

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

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



$$\frac{\partial f}{\partial z}$$

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

Example On Simple Function

gradients to maximize f (at a point)

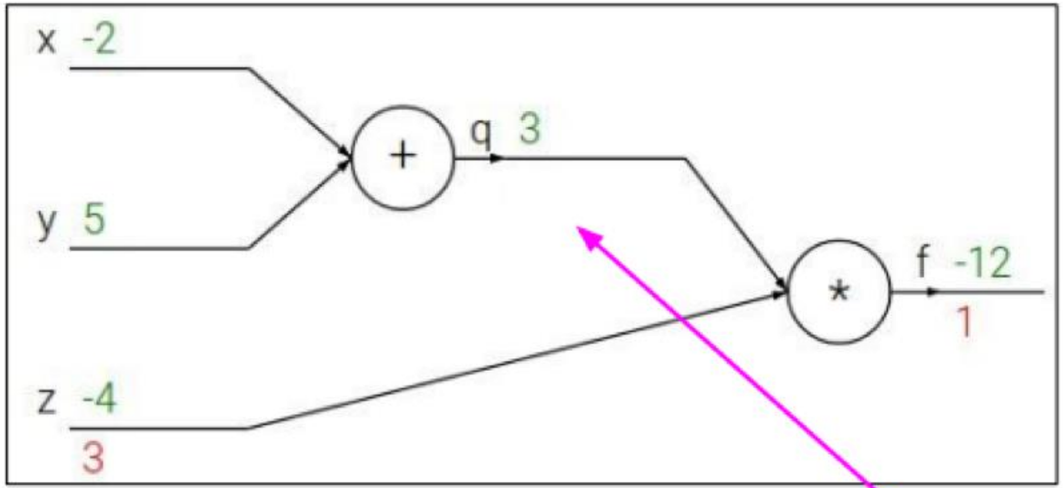
$$f(x, y, z) = (x + y)z$$

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

Node Gradients

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

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



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

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

http://cs231n.stanford.edu/slides/2020/lecture_4.pdf

Example On Simple Function

gradients to maximize f (at a point)

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

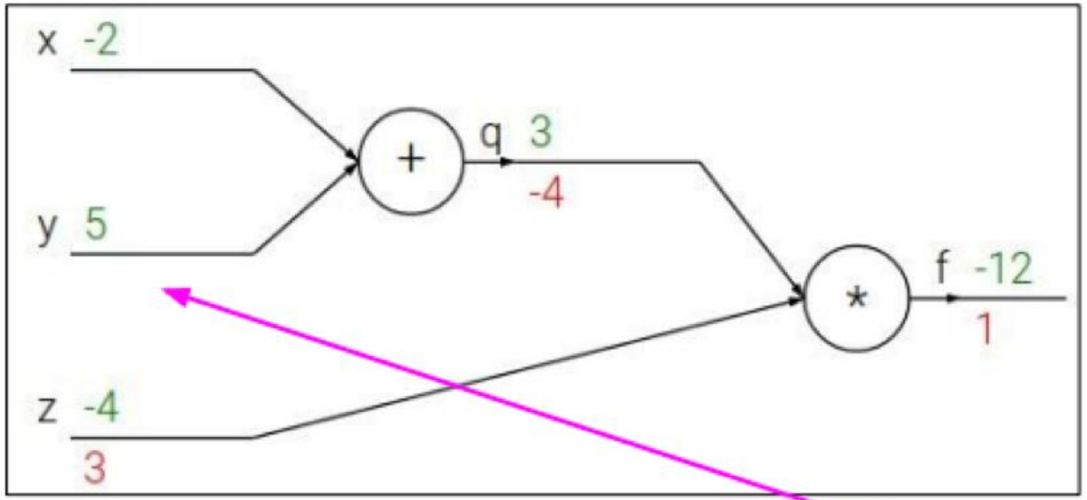
e.g. x = -2, y = 5, z = -4

Node Gradients

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

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

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



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

Chain rule:

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

Upstream gradient

Local gradient

http://cs231n.stanford.edu/slides/2020/lecture_4.pdf

Example On Simple Function

gradients to maximize f (at a point)

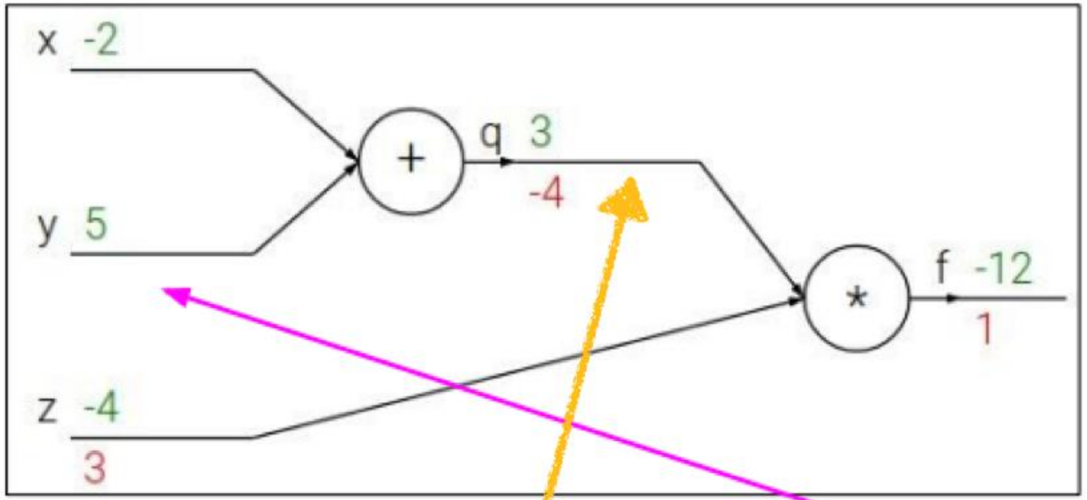
$$f(x, y, z) = (x + y)z$$

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

Node Gradients

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

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



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

Chain rule:

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

Upstream gradient

Local gradient

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

http://cs231n.stanford.edu/slides/2020/lecture_4.pdf

Example On Simple Function

gradients to maximize f (at a point)

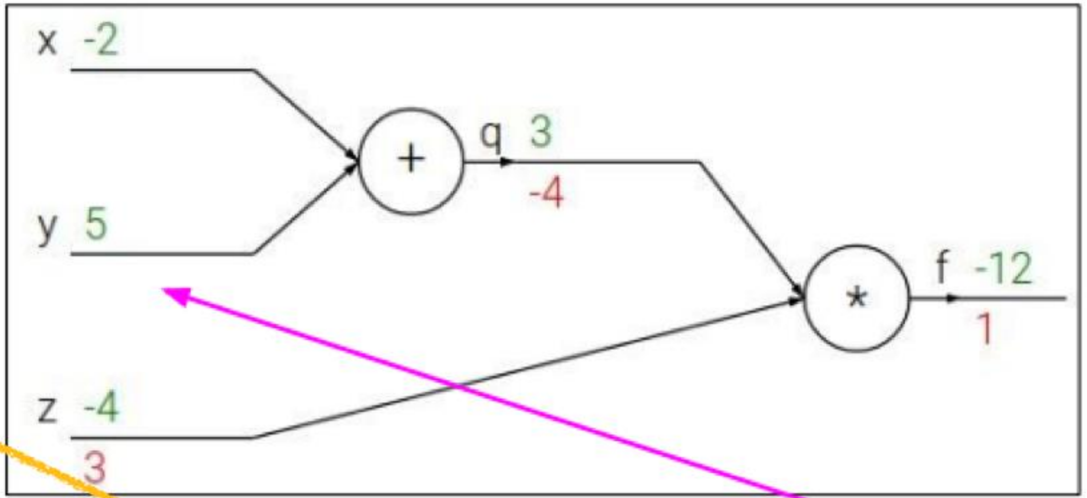
$$f(x, y, z) = (x + y)z$$

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

Node Gradients

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

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



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

Chain rule:

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

Upstream gradient

Local gradient

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

Example On Simple Function

gradients to maximize f (at a point)

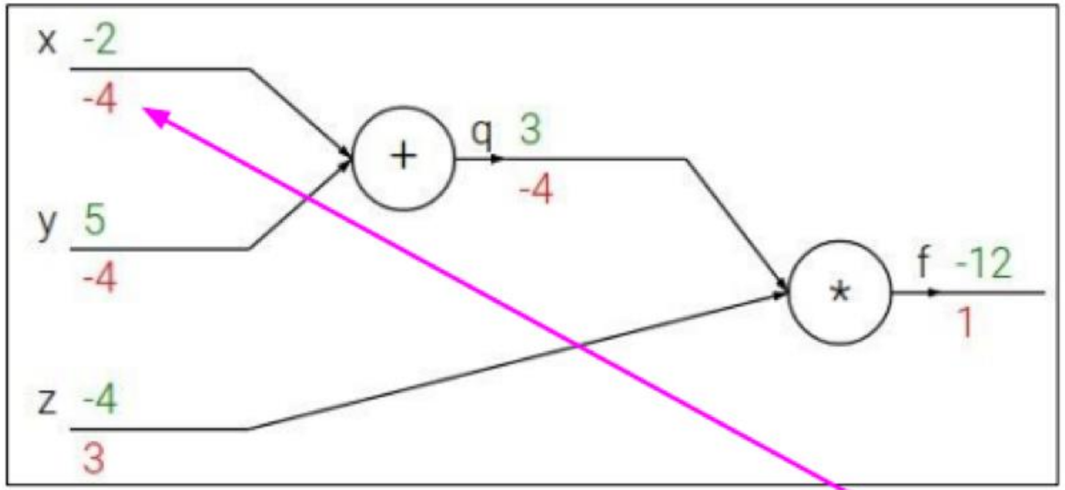
$$f(x, y, z) = (x + y)z$$

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

Node Gradients

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

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



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

Chain rule:

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

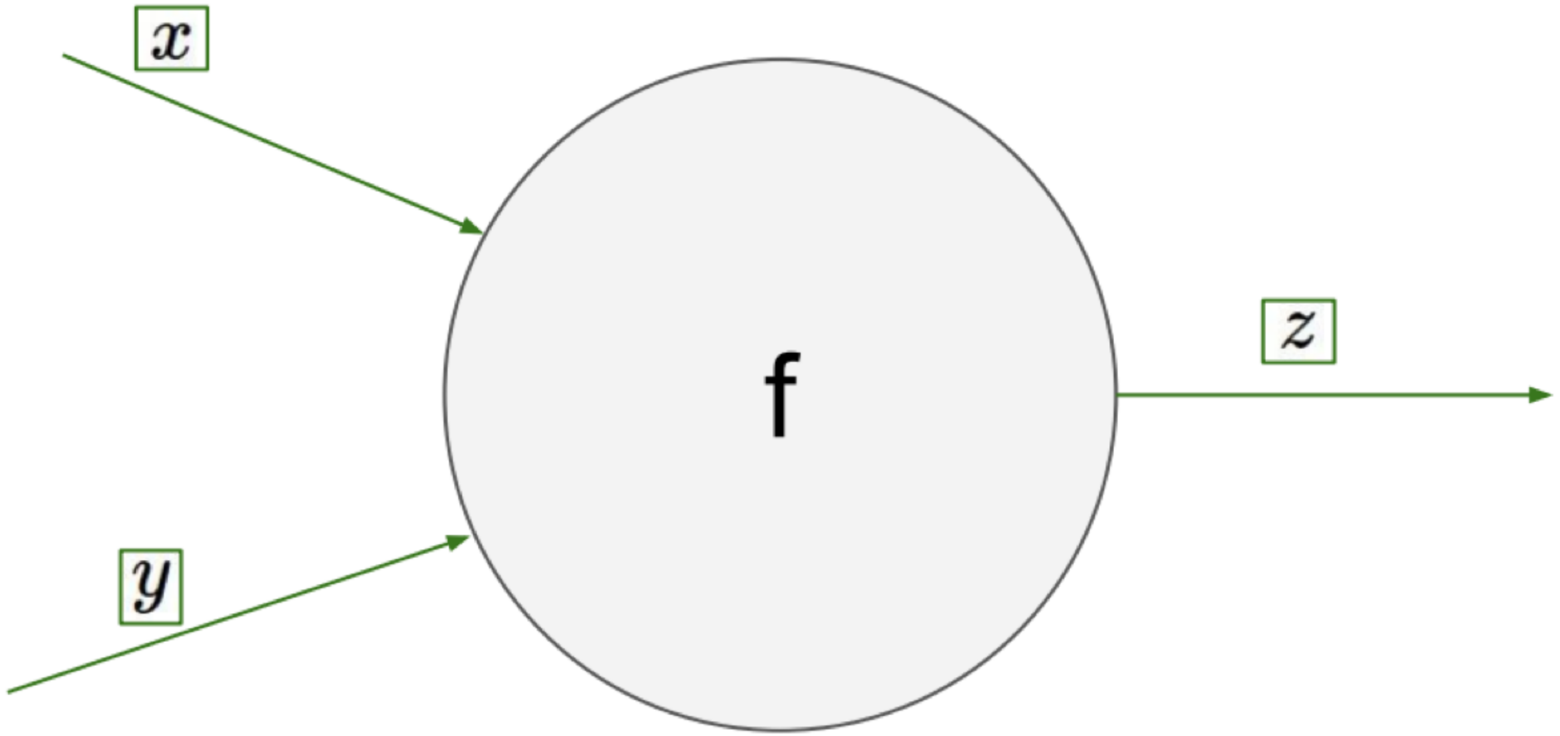
Upstream gradient

Local gradient

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

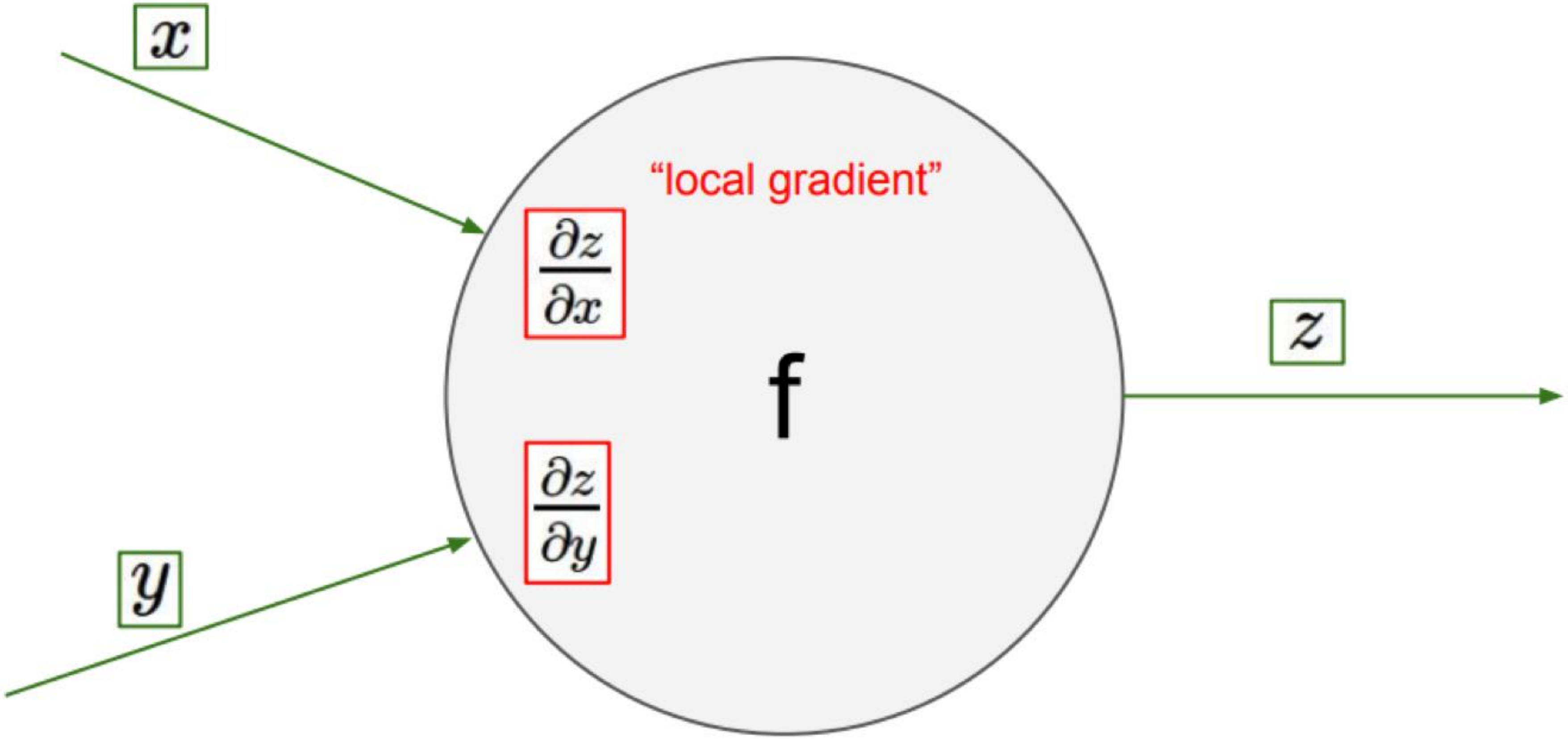
http://cs231n.stanford.edu/slides/2020/lecture_4.pdf

General Rule



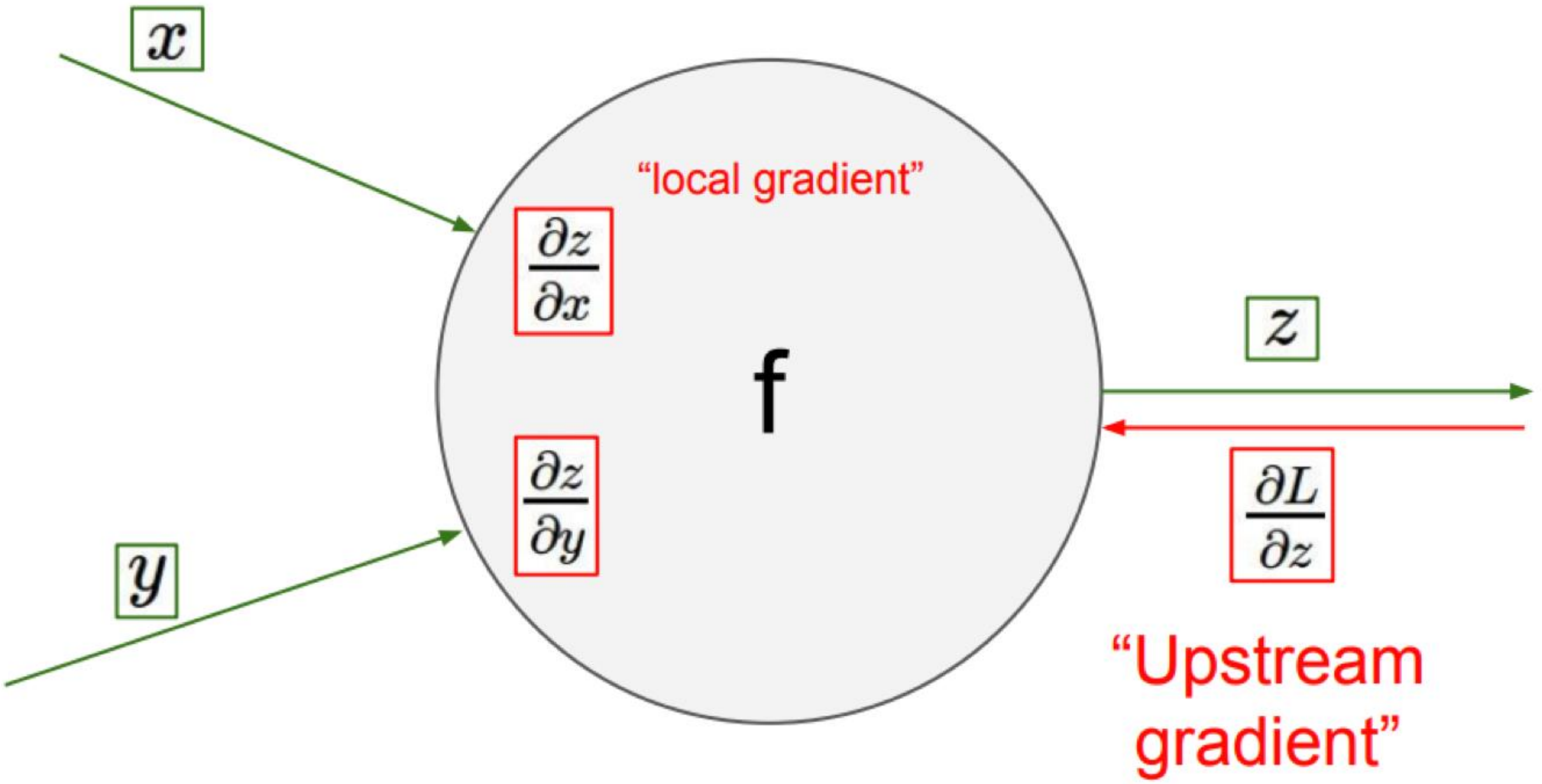
http://cs231n.stanford.edu/slides/2020/lecture_4.pdf

General Rule



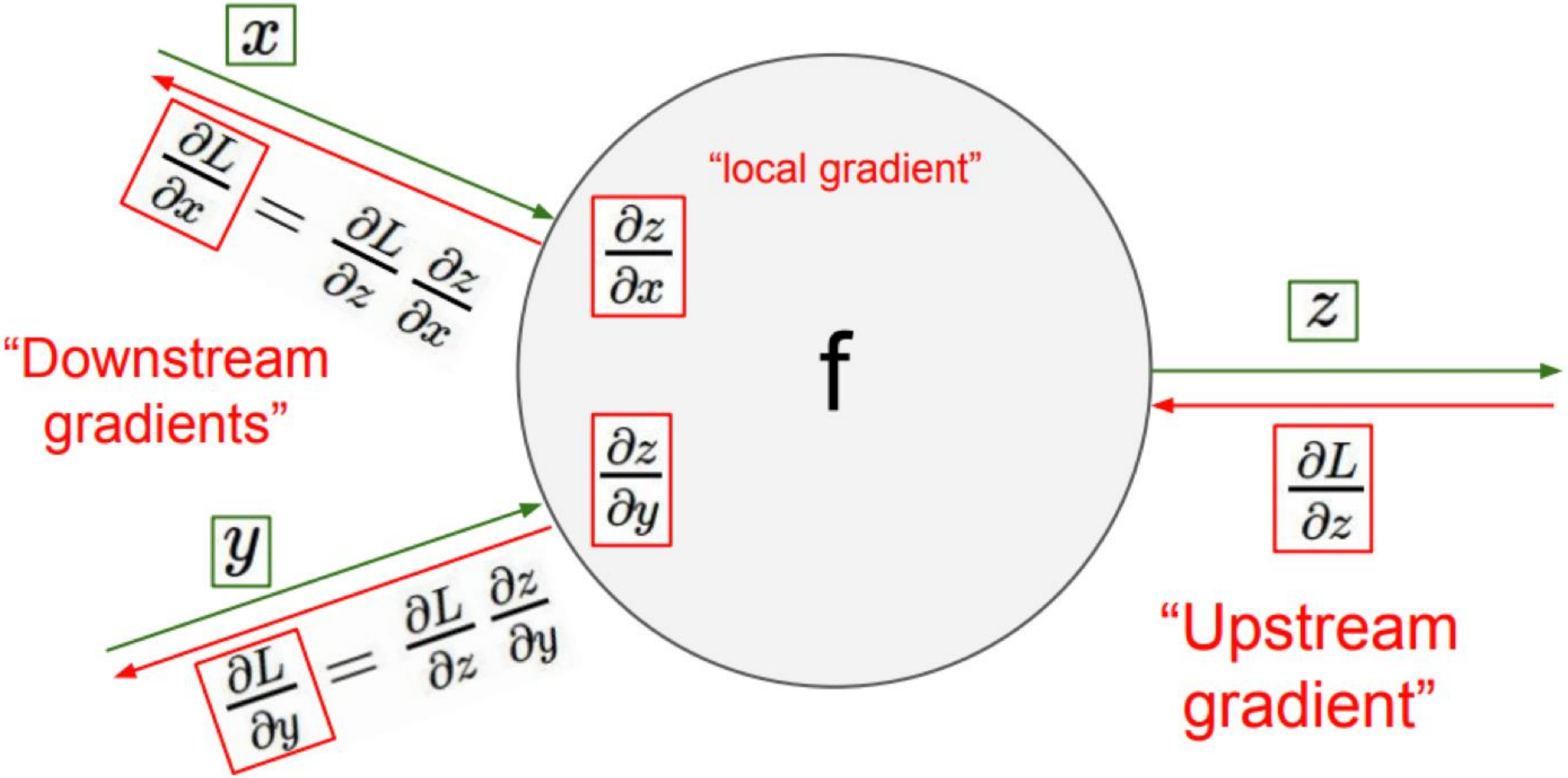
http://cs231n.stanford.edu/slides/2020/lecture_4.pdf

General Rule



http://cs231n.stanford.edu/slides/2020/lecture_4.pdf

General Rule



If x and y are inputs and parameters, we are done.
Otherwise, continue propagating gradients backward

http://cs231n.stanford.edu/slides/2020/lecture_4.pdf

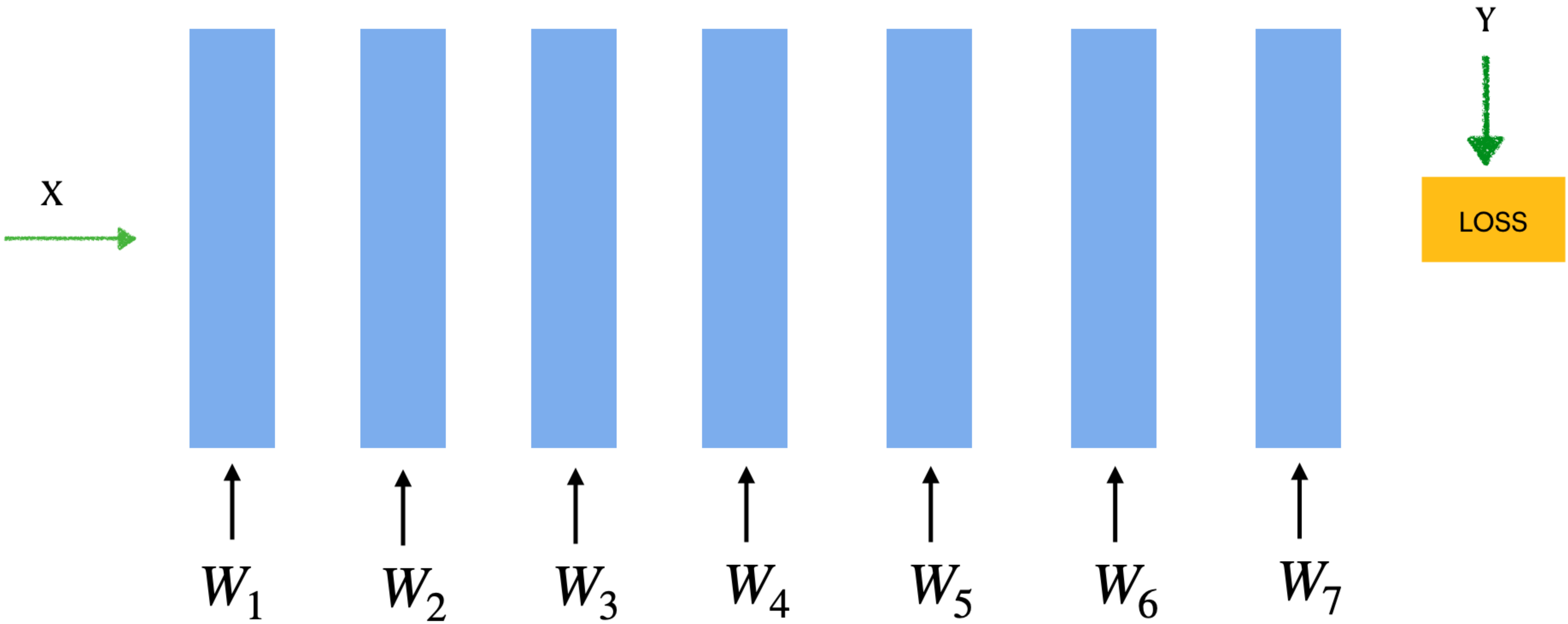
Backpropagation in general computational graph

- Forward propagation: visit nodes in topological sort order
  - Compute value of node given predecessors
- Backward propagagation:
  - Initialize output gradient as 1
  - Visit nodes in reverse order and compute gradient wrt each node using gradient wrt successors

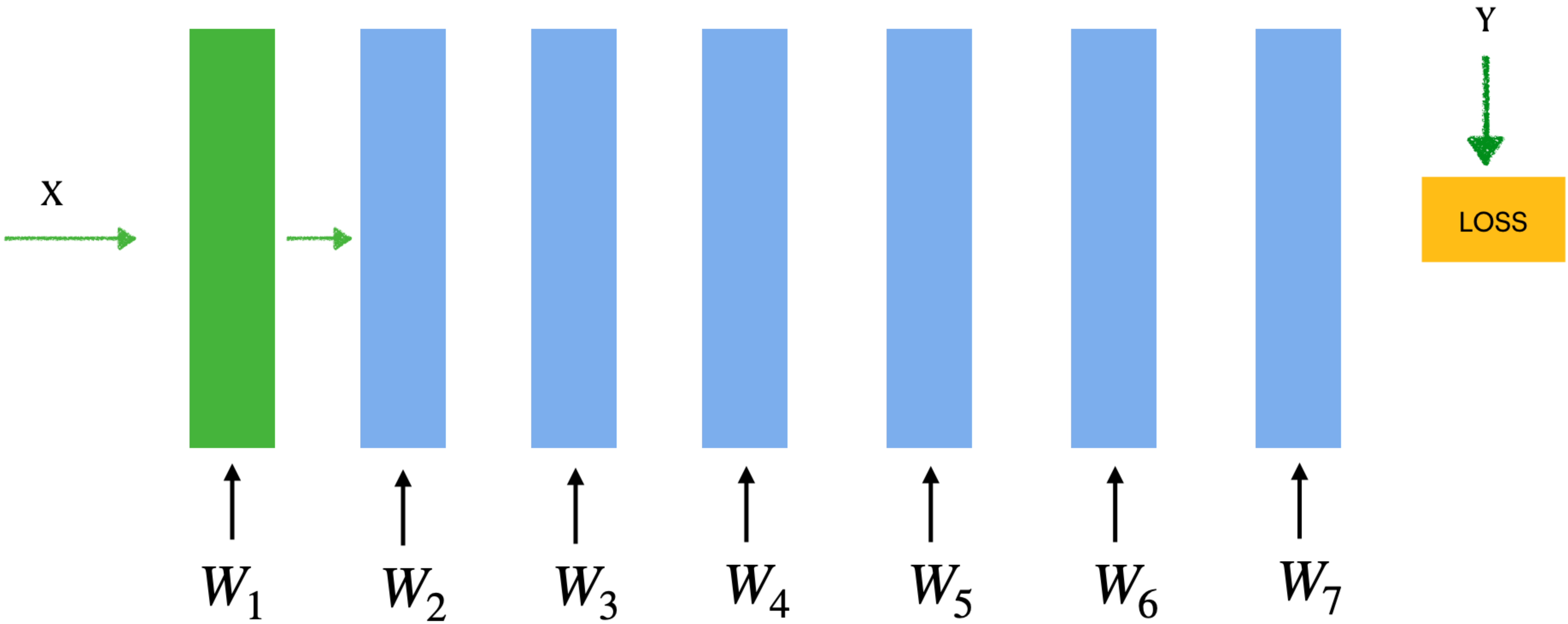
$$\frac{\partial L}{\partial x} = \sum_{i=1}^{n} \frac{\partial \mathcal{L}}{\partial y_i} \frac{\partial y_i}{\partial x}$$

$$\{y_1, \ldots, y_n\} = \text{ successors of } x$$

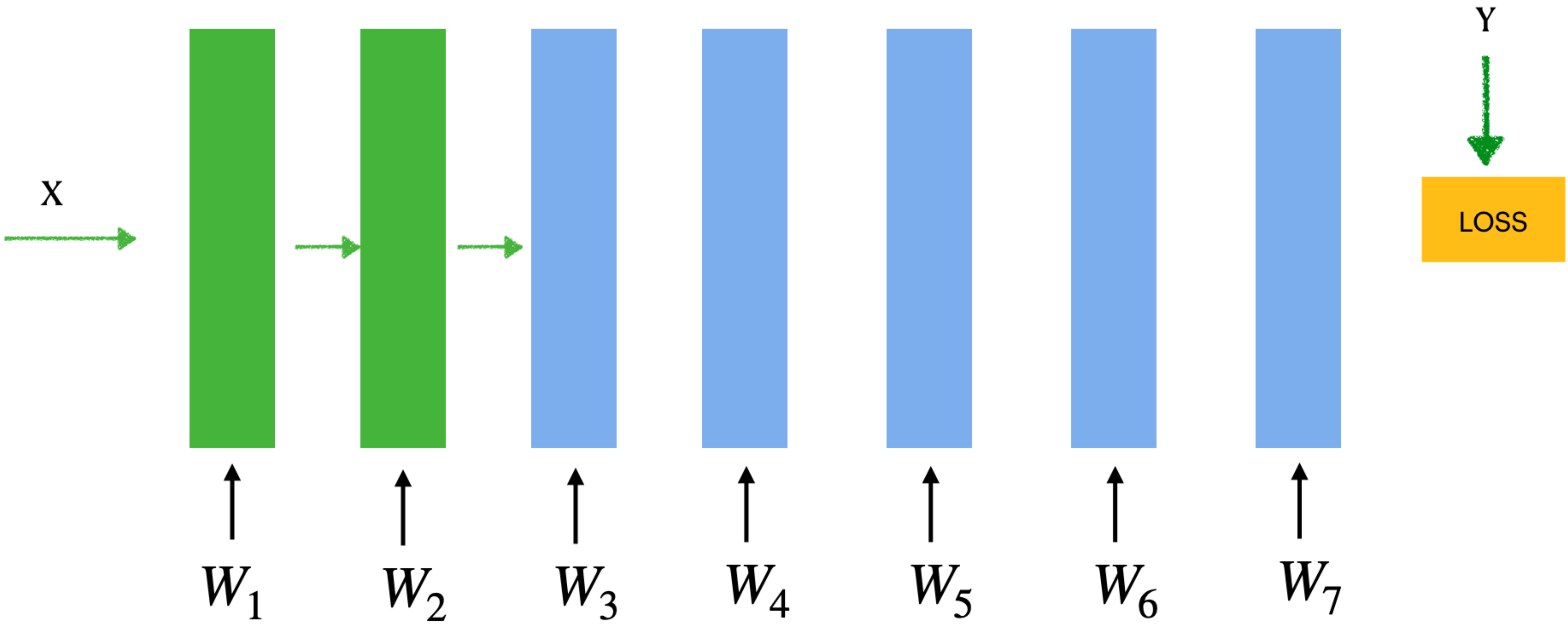
Forward Computation



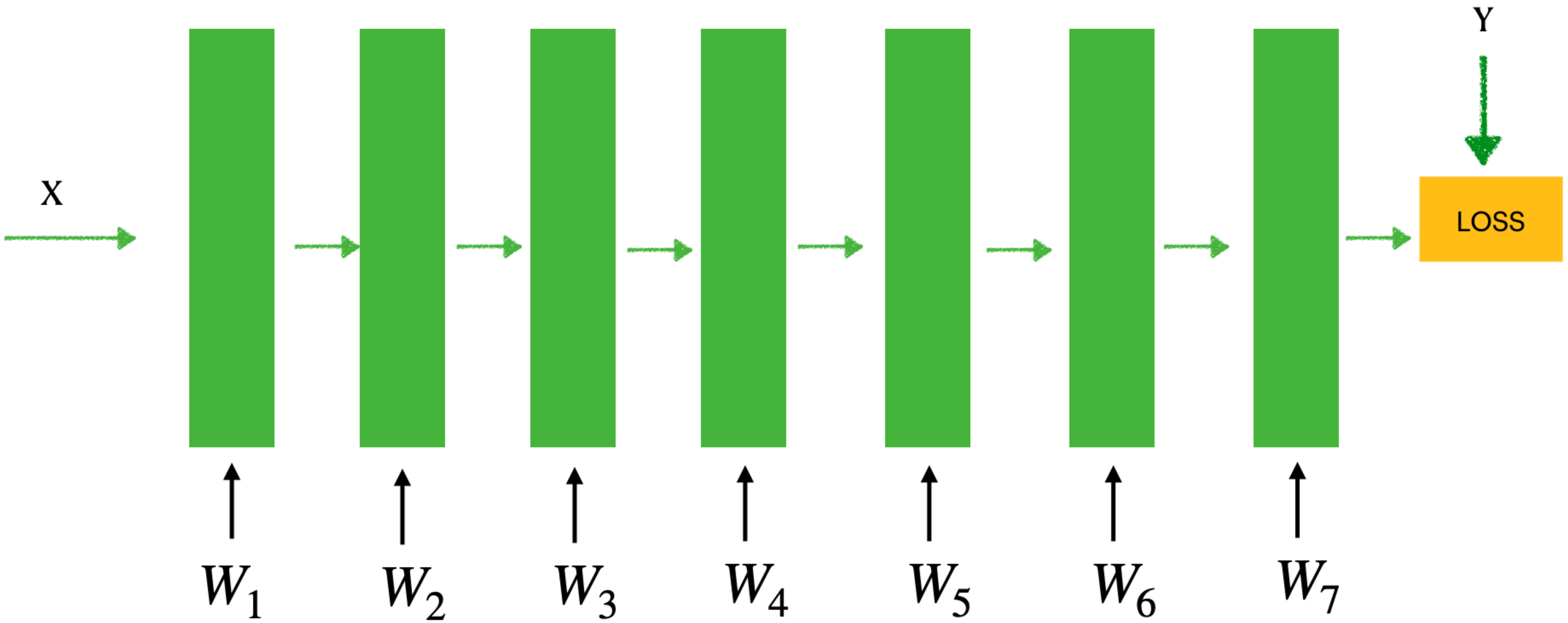
Forward Computation



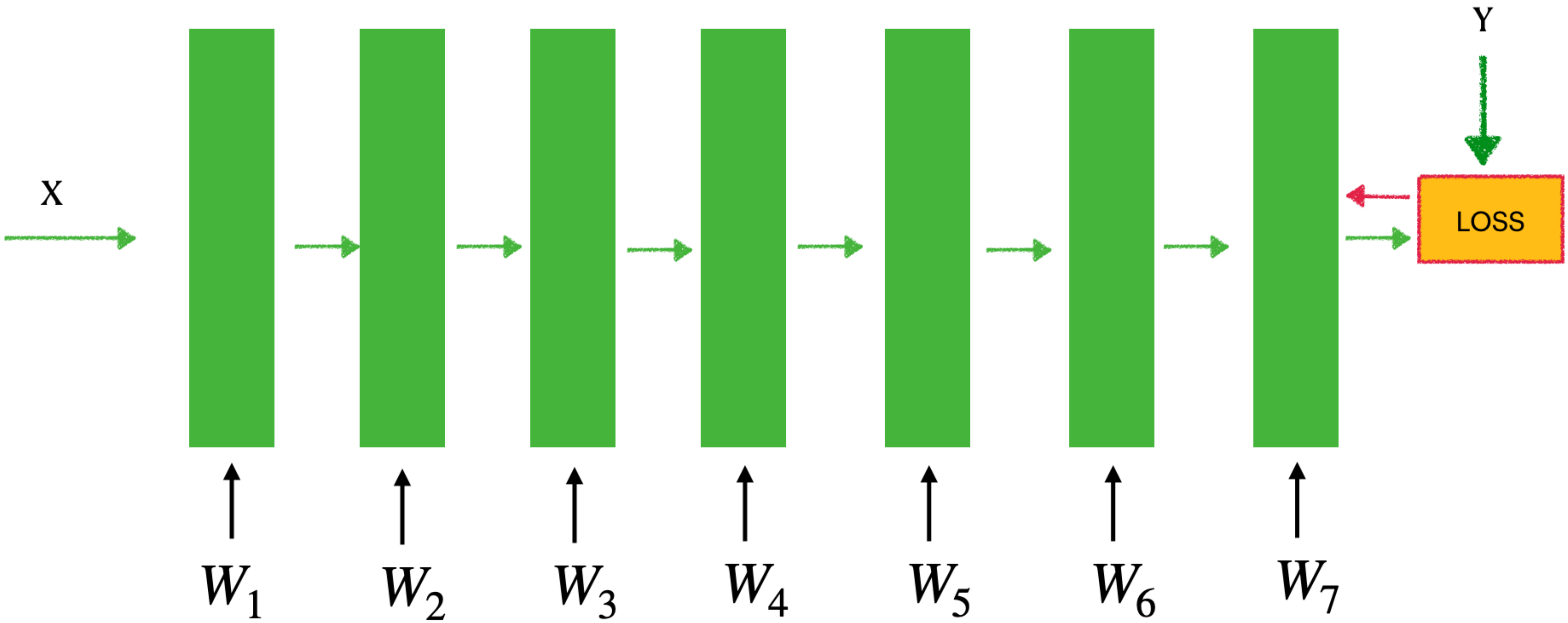
Forward Computation



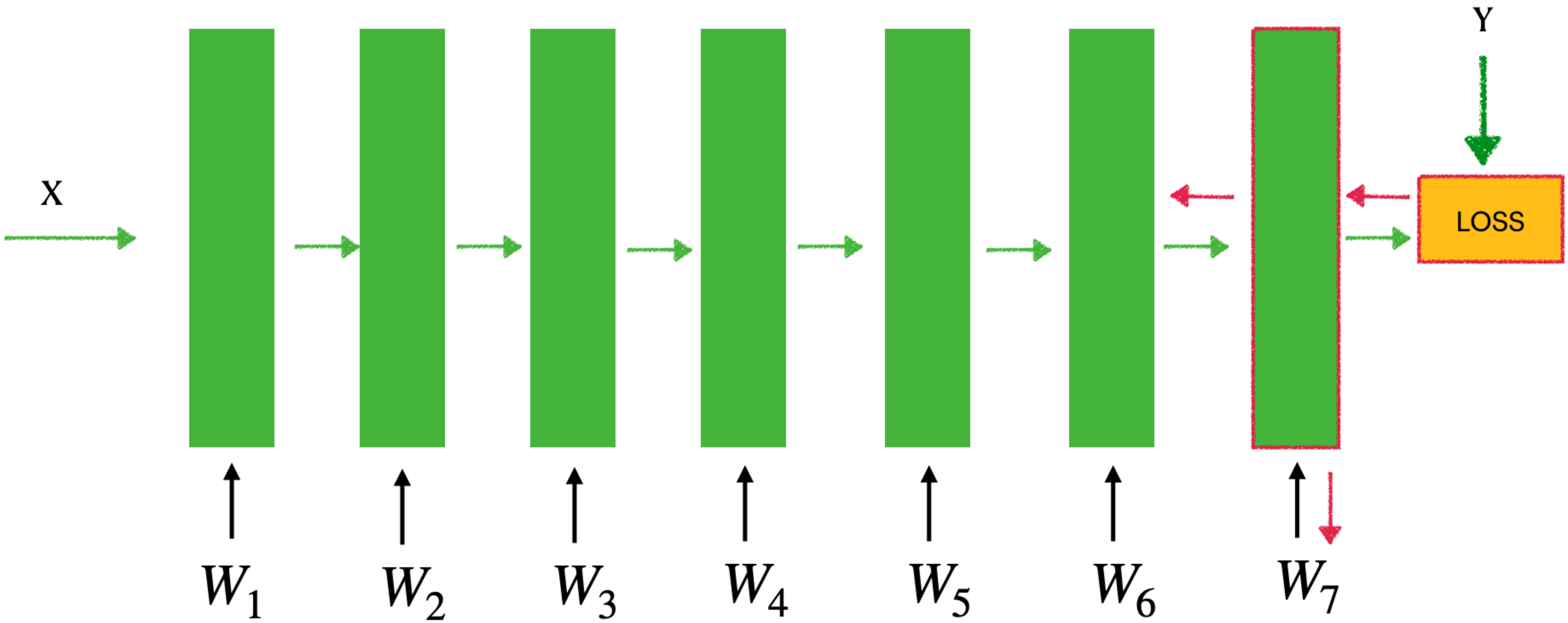
Forward Computation



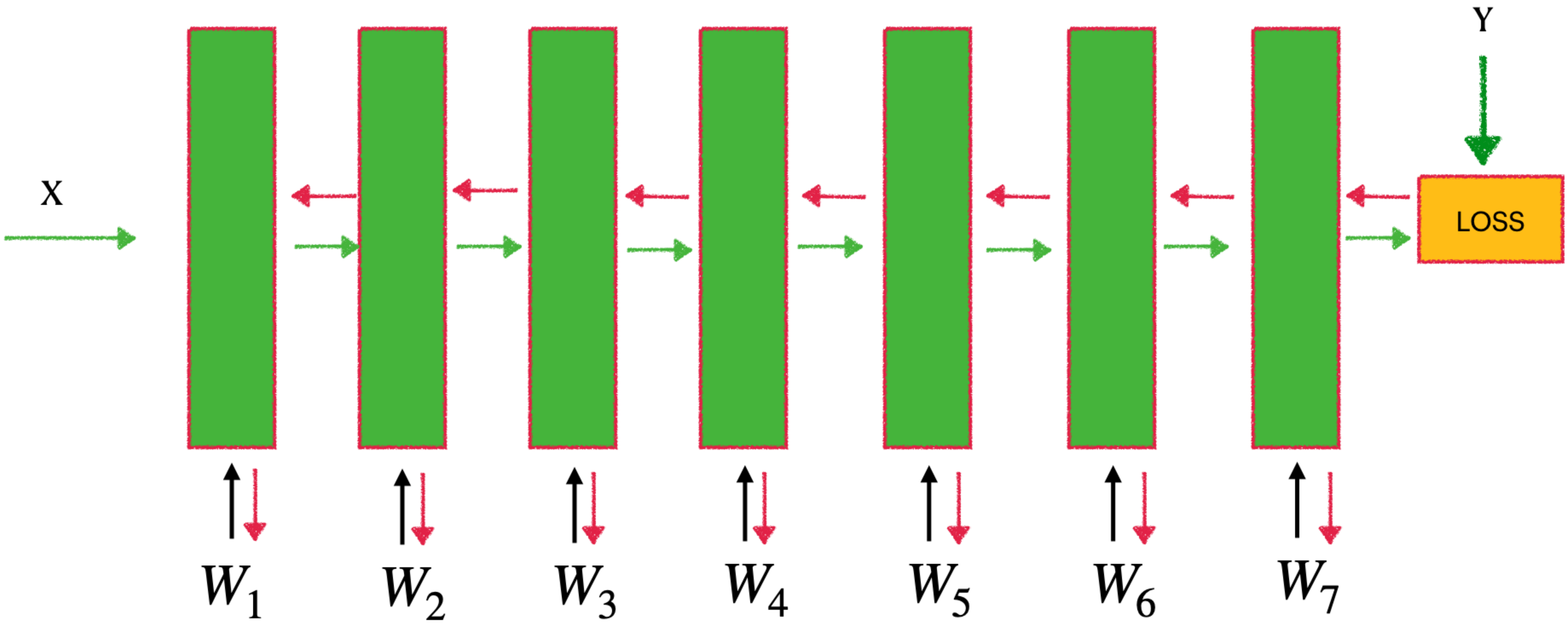
Backward Computation



Backward Computation



Backward Computation



Common Frameworks Do this Automatically

All you have to do is define the forward pass, in terms of known operators.

TensorFlow

https://www.tensorflow.org/

PyTorch

https://pytorch.org/

Today’s Lecture

- Model
  - Feedforward Neural Networks
- Loss functions
- Optimization
  - Stochastic Gradient Descent
  - Back-Propagation for Computing Gradients

Next Lecture

- Bag of tricks for optimization