

Lecture 7: Neural Networks (Part 1)

CIS 4190/5190

Fall 2024



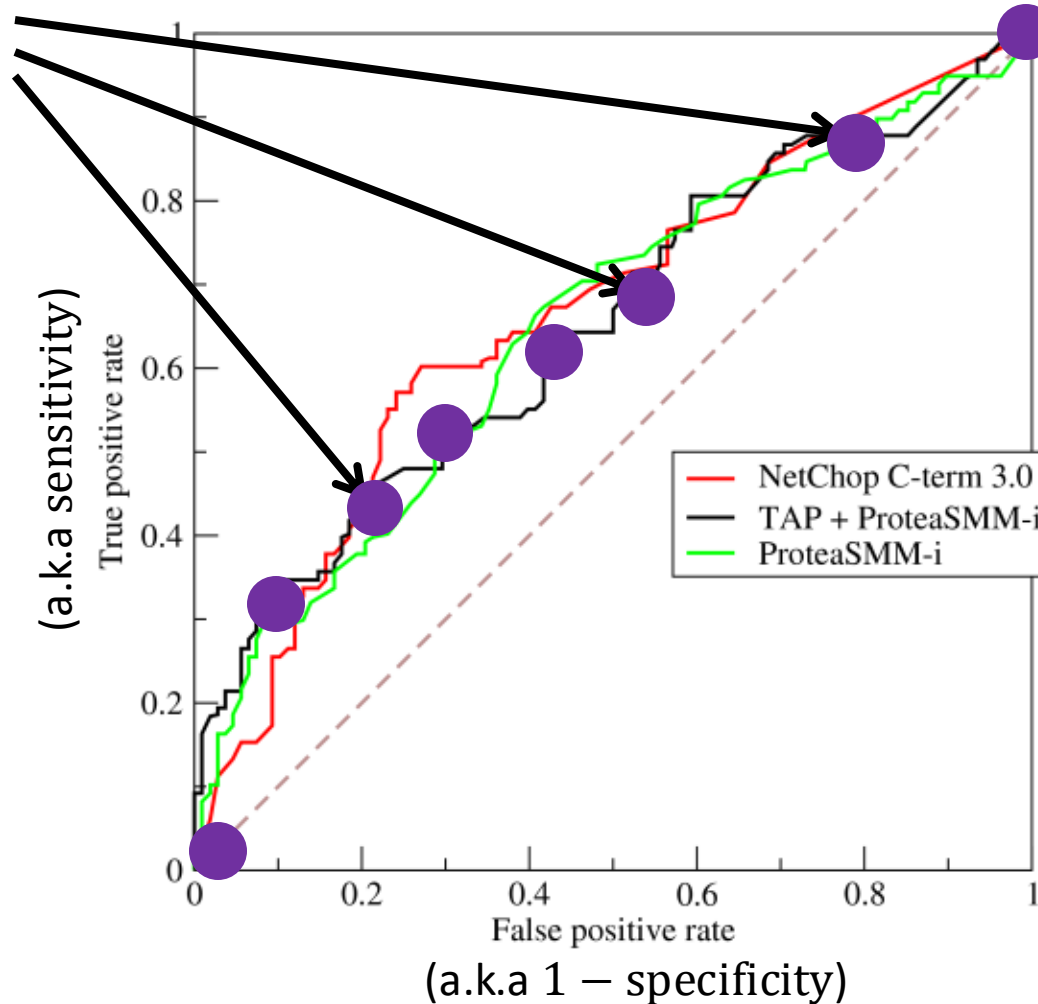
CIS 4190/5190: Lec 07 Part 1

Measuring Classification Performance (Part 2/2)

Visualization: ROC Curve

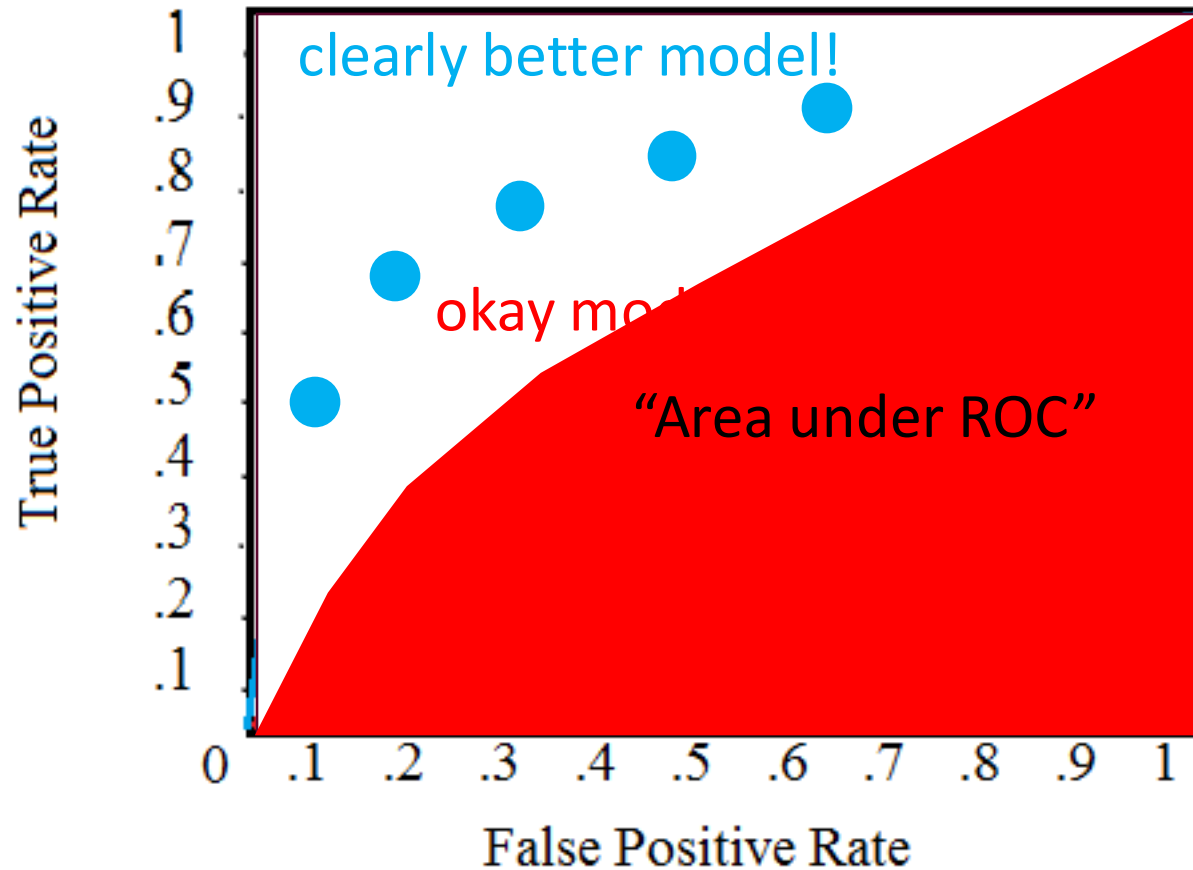
$$f_{\beta}(x) = 1(\beta^T x \geq \tau)$$

Each point on this curve corresponds to a choice of τ



(PS: ignore the red and green curves)

Compare To Random Performance on TPR-FPR ROC



Area under ROC curve is another single scalar metric that evaluates $\hat{\beta}(Z)$ in a way that is independent of the threshold τ .

Rationale: downstream user can tune τ relatively easily to fit their needs.

There is a similar notion of the Area under "Precision Recall" curve. Also known as "average precision."

Summarizing: Optimizing Prediction Threshold

- Consider hyperparameter τ for the threshold:

$$f_{\beta}(x) = 1(\beta^{\top}x \geq \tau)$$

- Unlike most hyperparameters, we choose this one **after** we have already fit the model on the training data
 - Then, choose the value of τ that optimizes the desired metric
 - Pick τ using ideally validation data, but training data OK in a pinch

Optimizing a Classification Metric

- We are training a model to minimize NLL, but we have a different “true” metric that we actually want to optimize
- Two strategies (can be used together):
 - **Strategy 1: (After training)** Optimize prediction threshold
 - **Strategy 2: (Before training)** Upweight positive (or negative) examples

Class Re-Weighting

- **Weighted NLL:** Include a class-dependent weight w_y :

$$\ell(\beta; Z) = - \sum_{i=1}^n w_{y_i} \cdot \log p_{\beta}(y_i | x_i)$$

- A “softened” version of the weighted mean of per-class accuracy that is amenable to optimize
- **Intuition:** Tradeoff between accuracy on negative/positive examples
 - To improve sensitivity (true positive rate), upweight positive examples
 - To improve specificity (true negative rate), upweight negative examples
- Can use this strategy to learn β , and the first strategy to choose τ

Classification Metrics

- NLL isn't usually the “true” metric
 - Instead, frequently used due to good computational properties
- Many choices with different meanings
- Typical strategy:
 - Learn β by minimizing the NLL loss
 - Choose class weights w_y and threshold τ to optimize desired metric

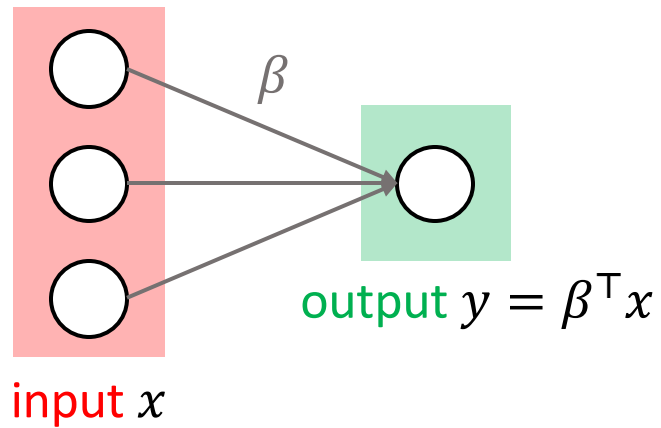
Lecture 7: Neural Networks (Part 1)

CIS 4190/5190

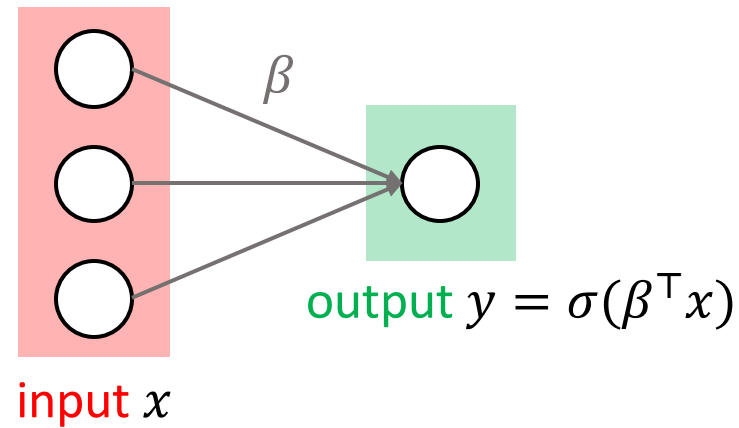
Fall 2024

So far in this class

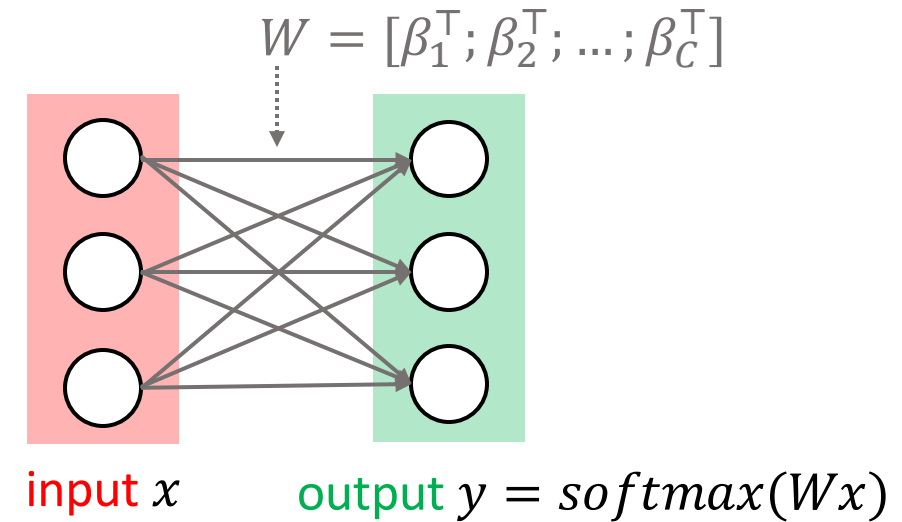
Linear Regression



Binary Classification



Multi-Class Classification

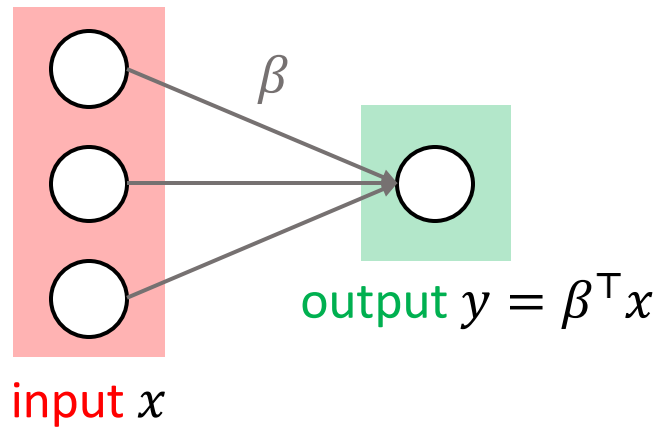


$$p_W(y = c | x) = \frac{e^{\beta_c^T x}}{\sum_{y'} e^{\beta_{y'}^T x}}$$

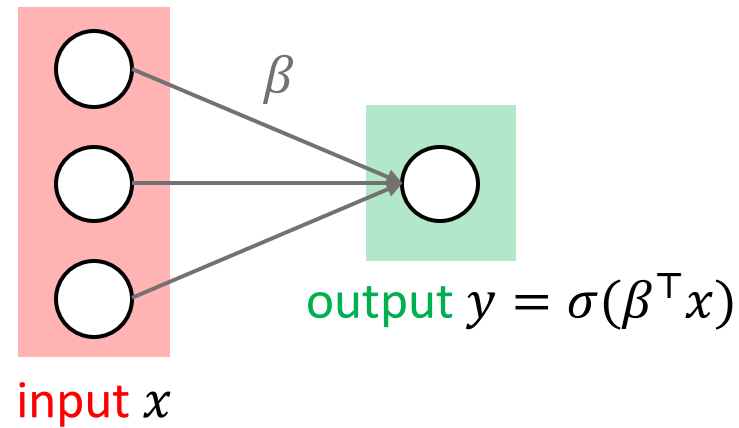
A Unifying View

Linear transformation of input features followed by an activation function

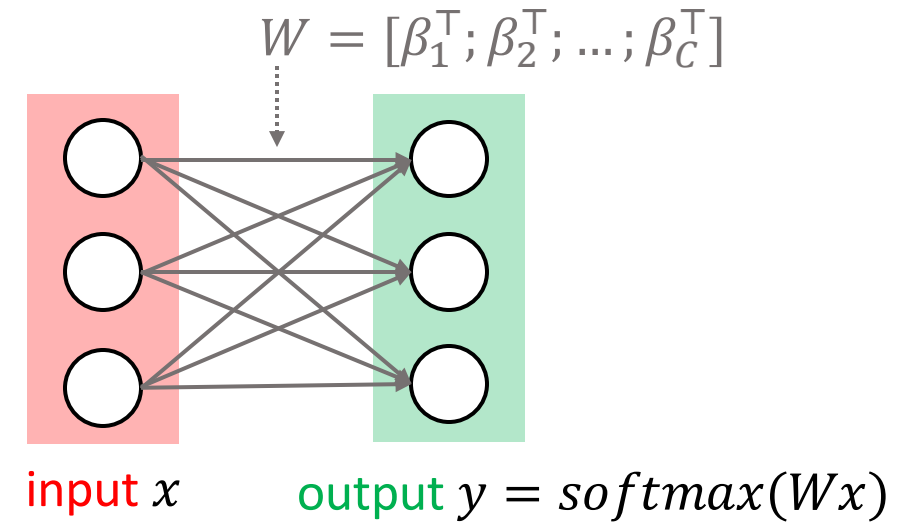
Linear Regression



Binary Classification



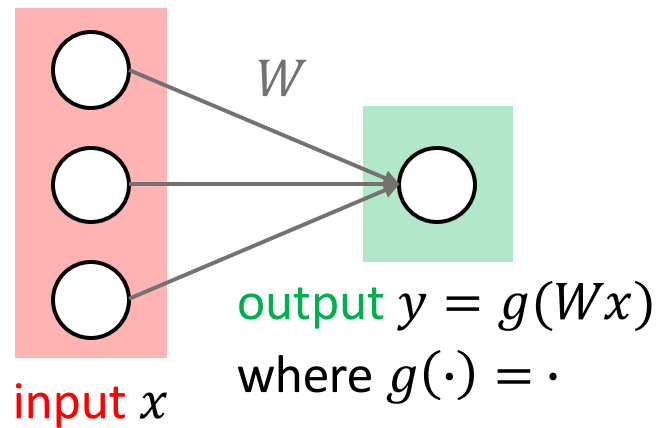
Multi-Class Classification



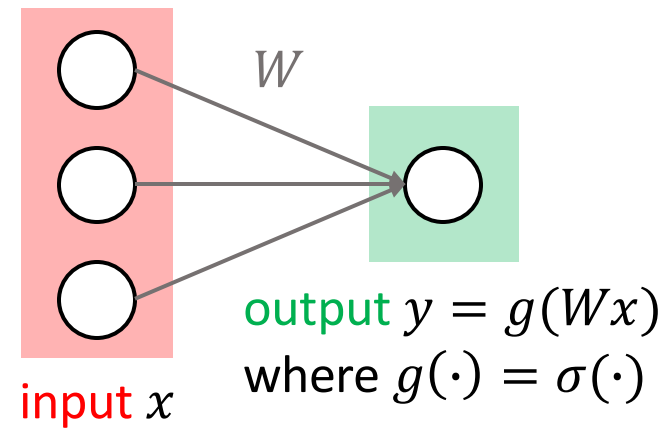
A Unifying View

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

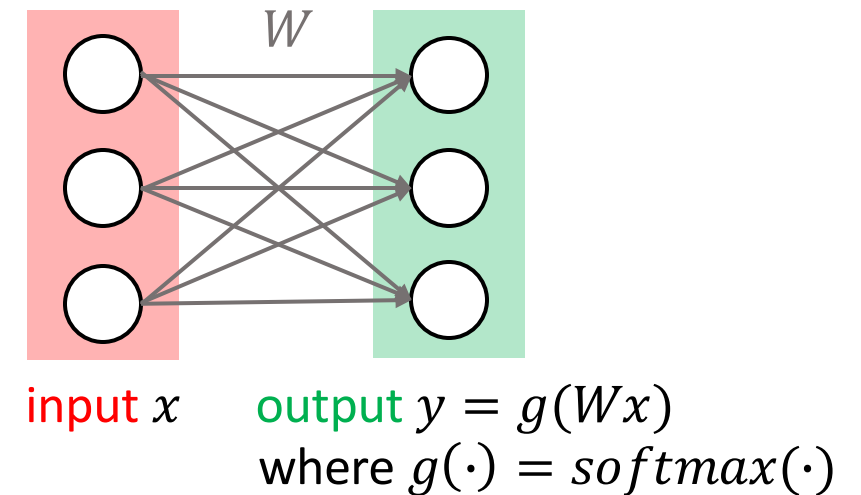
Linear Regression



Binary Classification

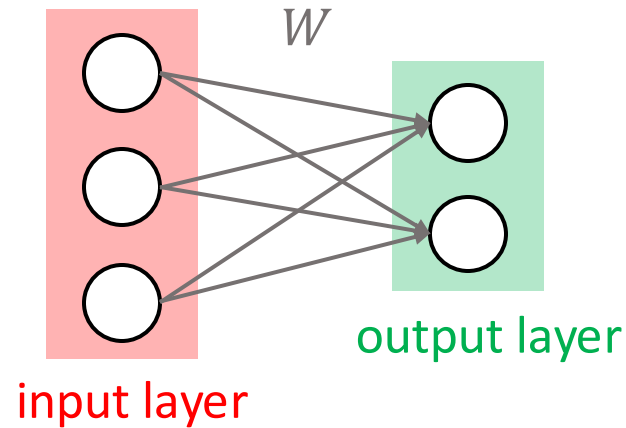


Multi-Class Classification



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

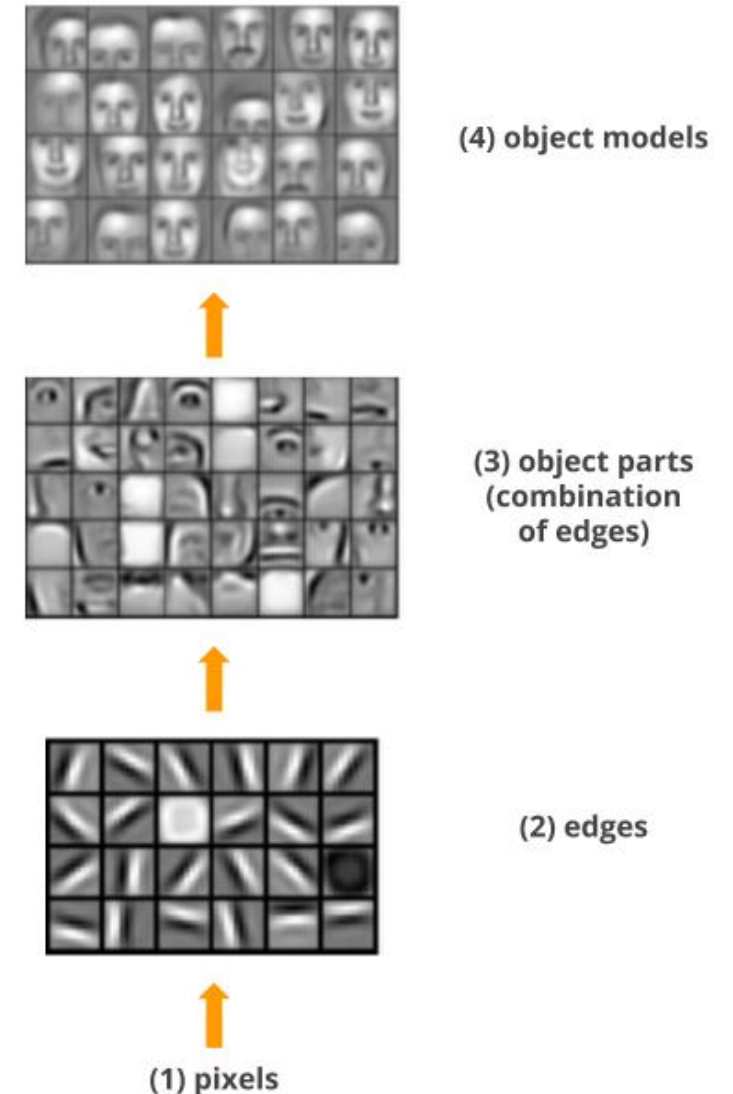
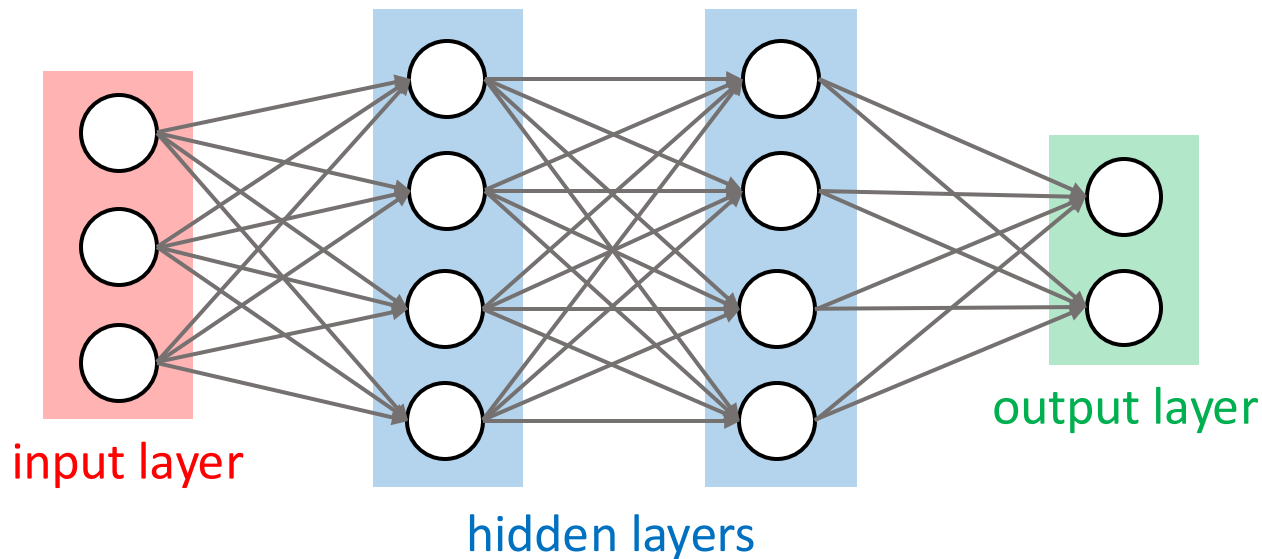
A Unifying View: Single-Layer Neural Network



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

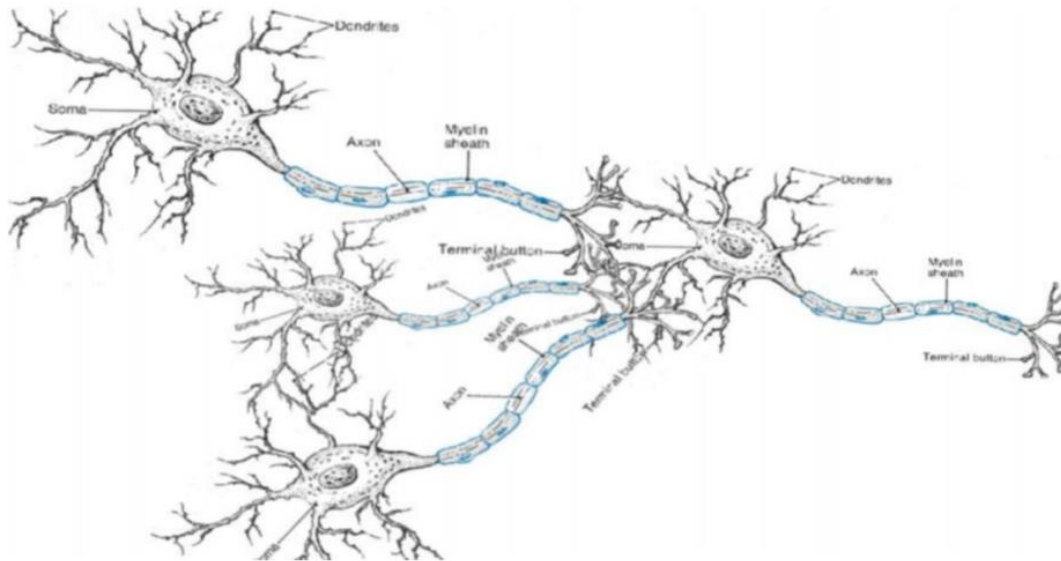
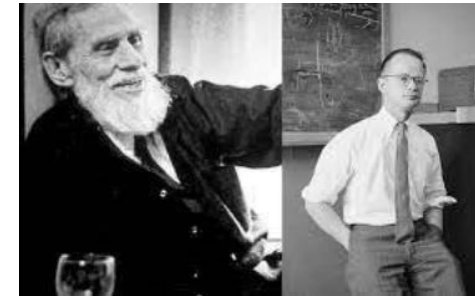
The “Promise” of Deep Neural Networks

- **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)
- **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

- Regularization like Dropout



Next lecture

- Some domain specific tricks & architectures

- Word embedding for NLP



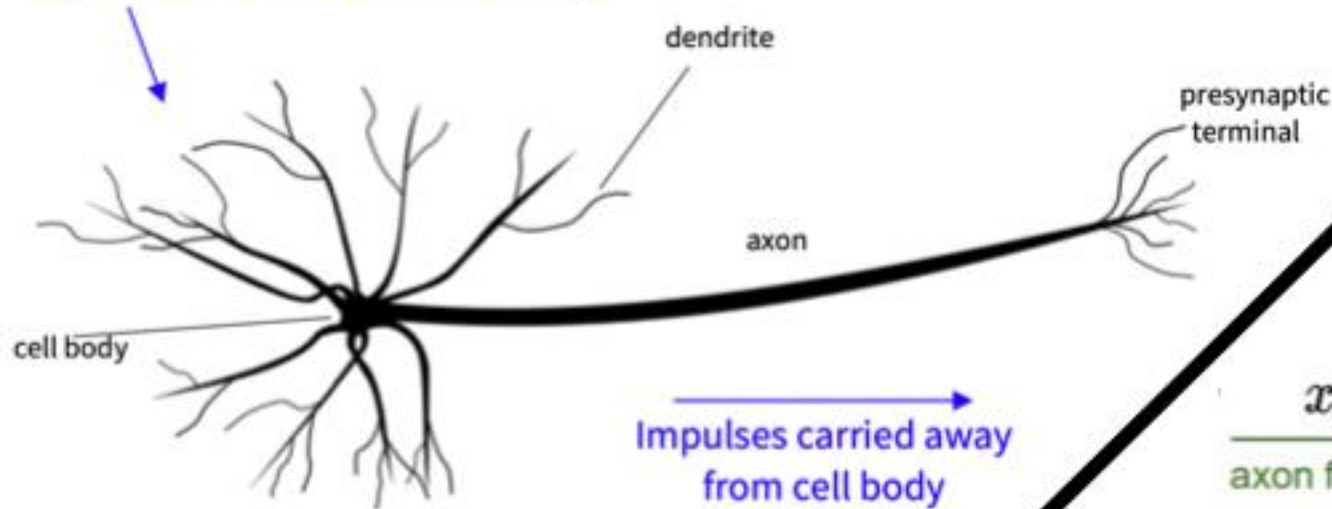
**Topics for 2nd half
of the semester**

Today's Lecture

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

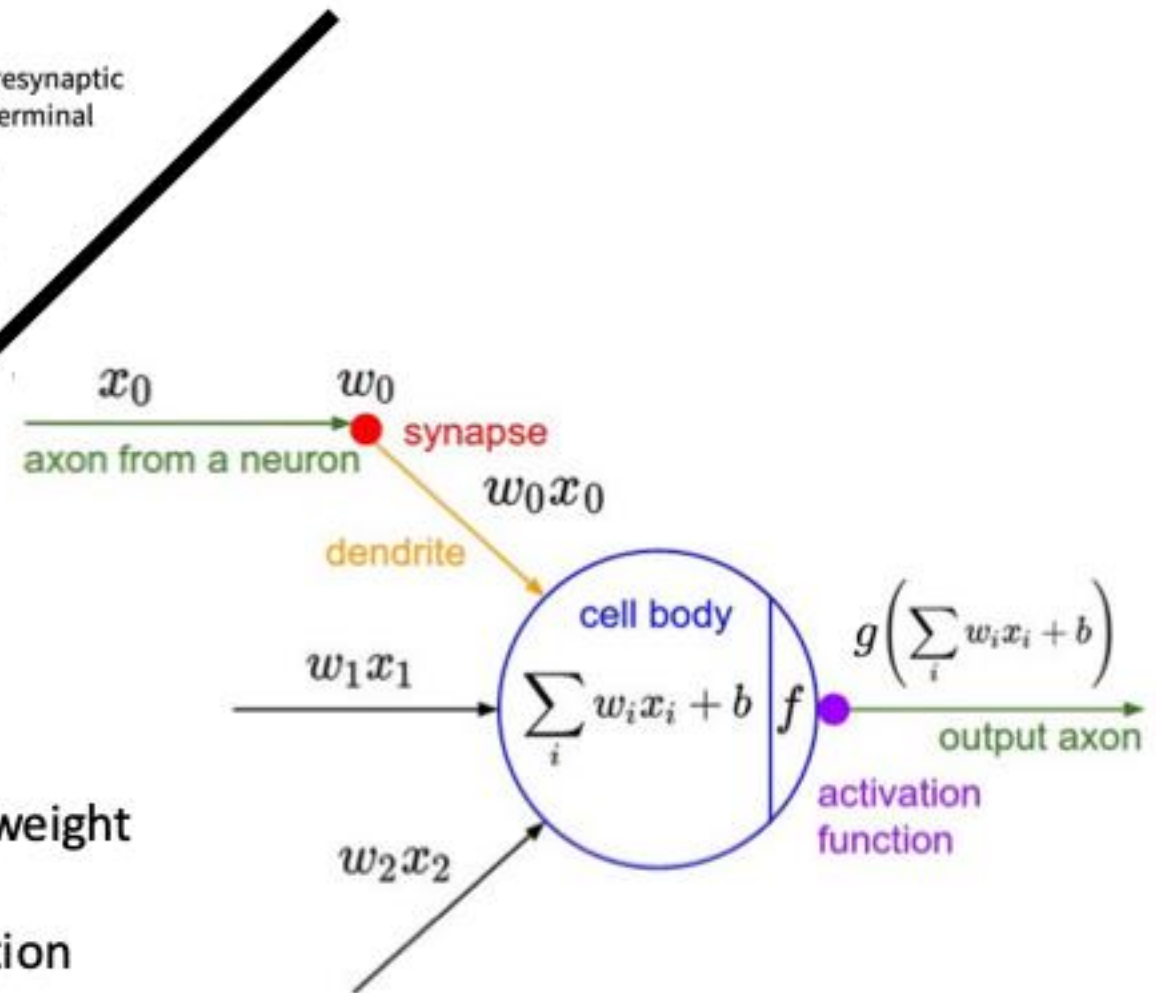
Simplified Brain Modeling

Impulses carried toward cell body



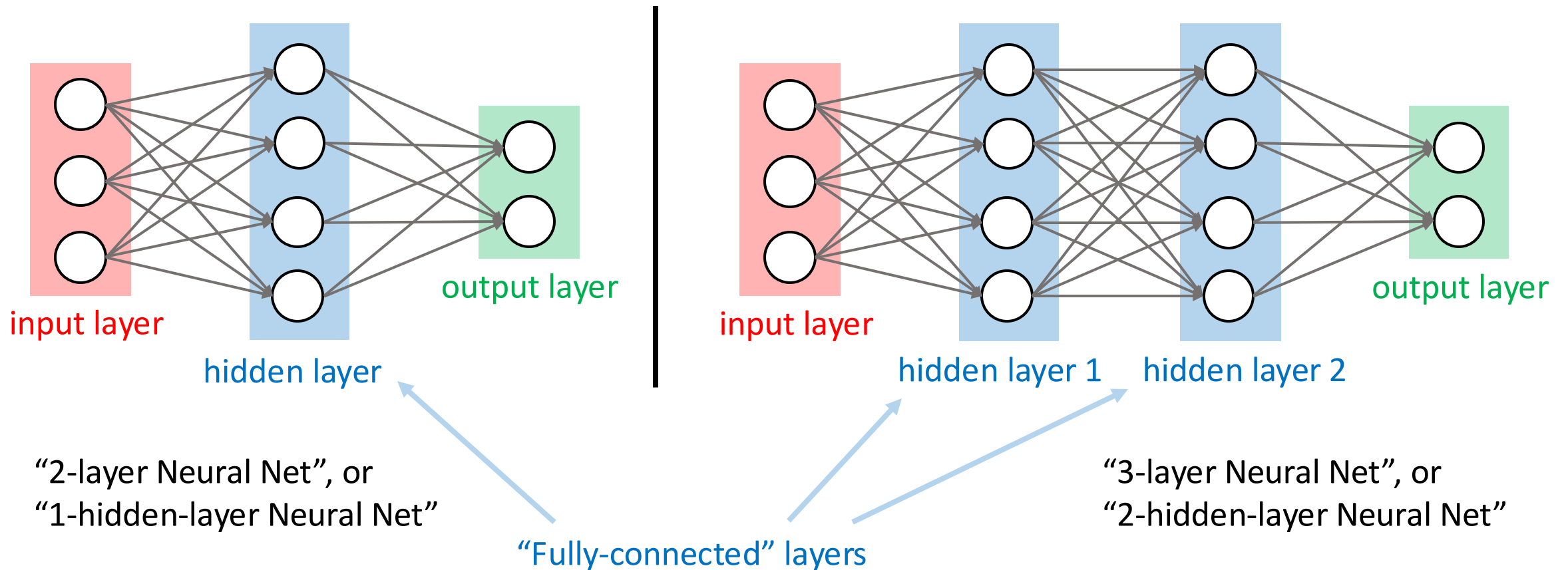
This image by Felipe Perucho is licensed under [CC BY 3.0](https://creativecommons.org/licenses/by/3.0/)

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: $y = W_3 g(W_2 g(W_1 x))$

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

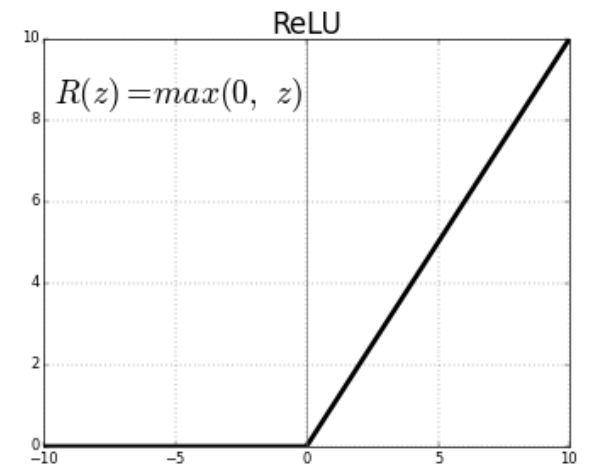
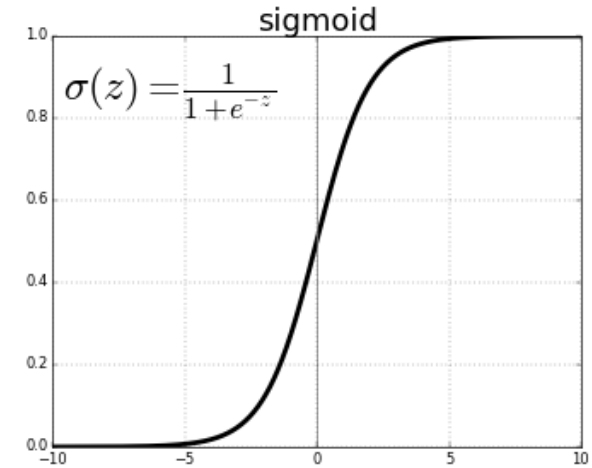
g is a non-linear activation function for hidden layers

(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: $y = W_2 g(W_1 x)$ \longrightarrow $y = W_2 W_1 x$

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

A: We would end up with linear classifier!

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

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: $\mathcal{L}_{\text{MSE}}(y, y^*) = (y - y^*)^2$

- 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 derivatives 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.

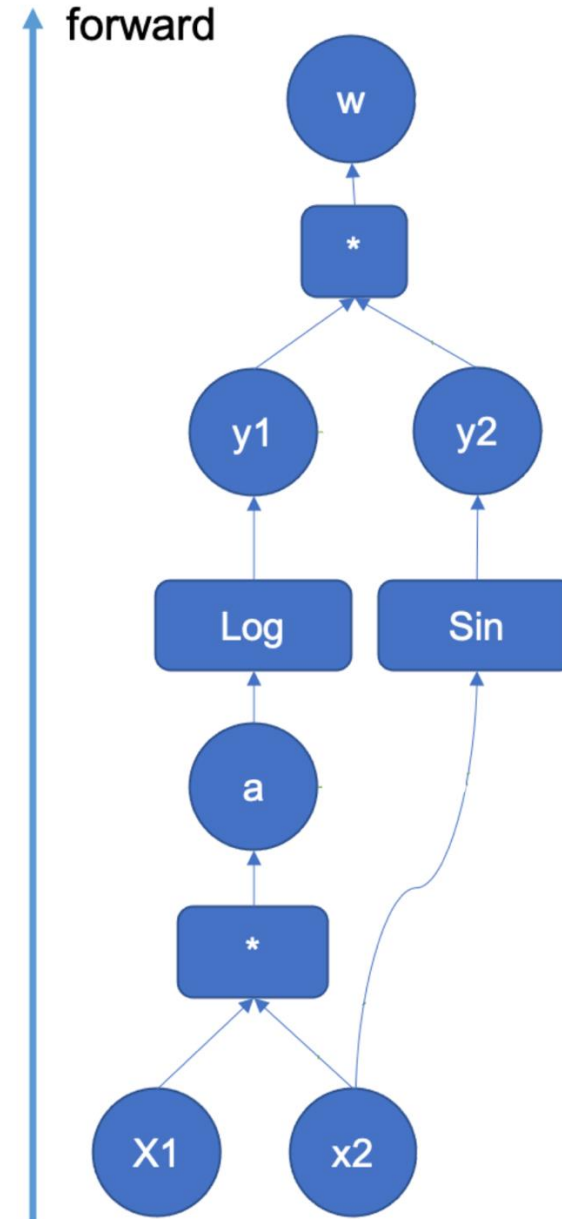


Computational Graph

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

- Consider the following function:

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



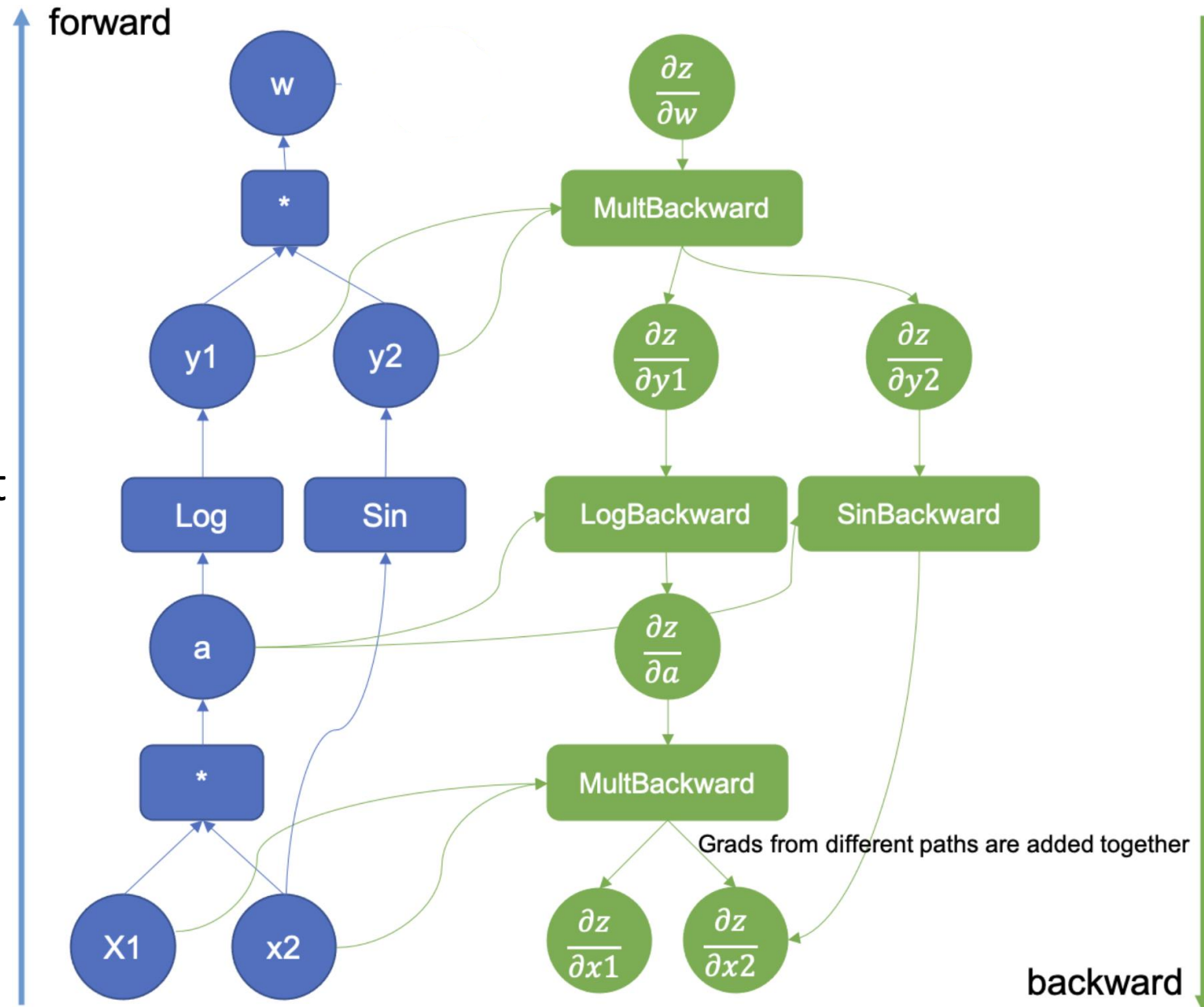
Full Graph

Requirement for each operator:

Forward: compute their output function given input.

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

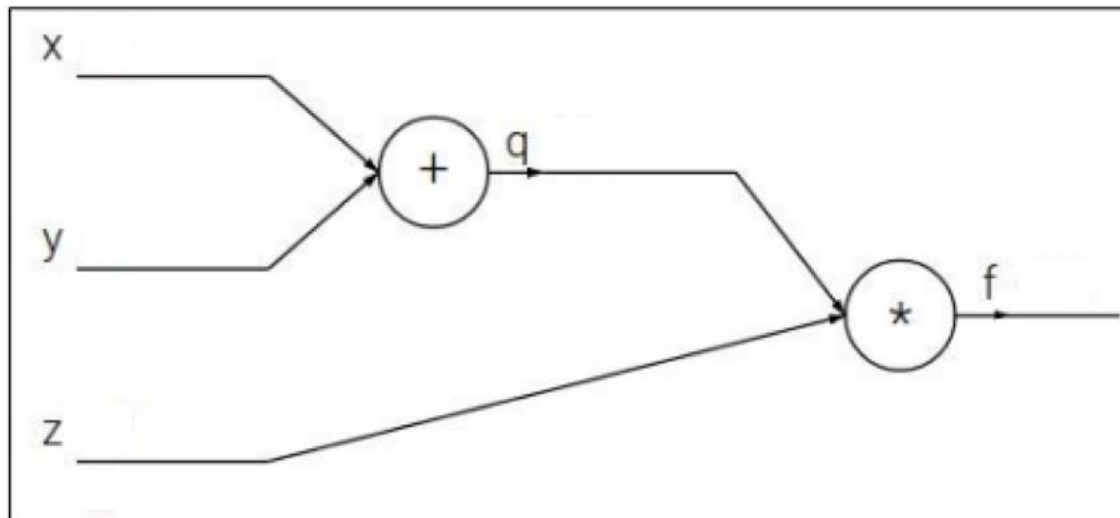
If all operators can do forward & backward computation, we can compute the derivative of the output with respect to any input, procedurally.



Example On Simple Function

gradients to maximize f

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

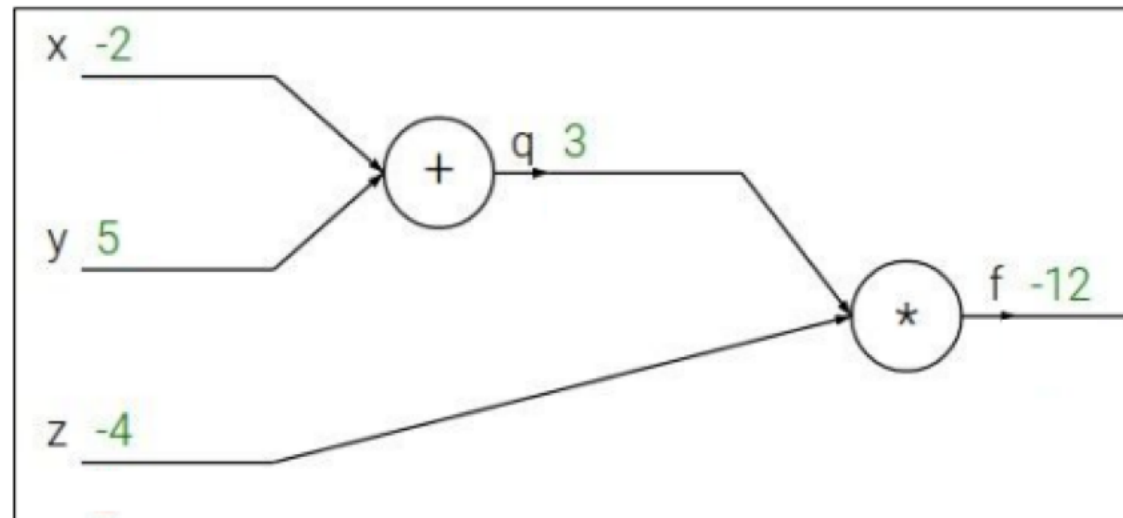


(circles represents operators here)

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)

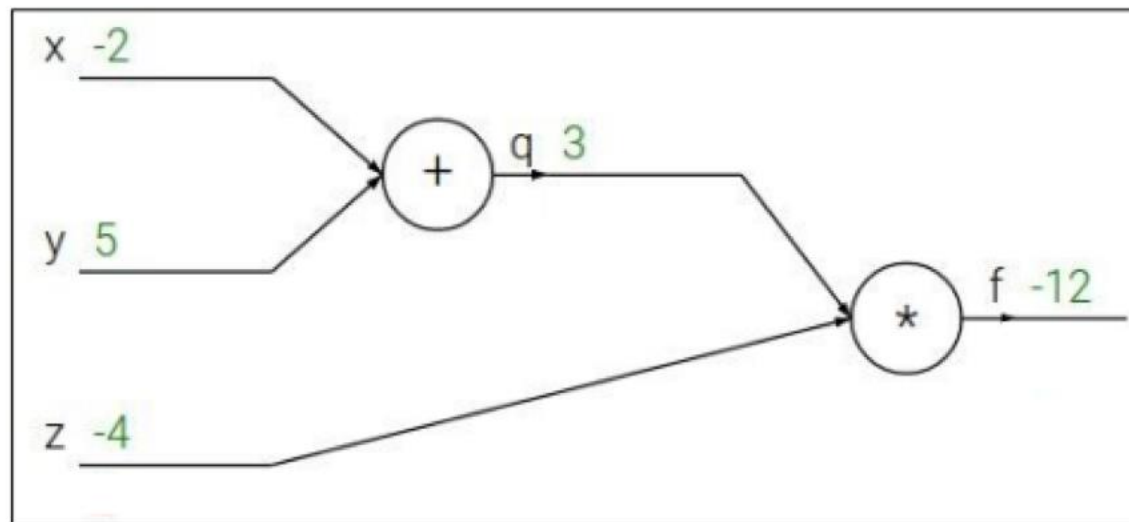
$$f(x, y, z) = (x + y)z$$

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

Node
Gradients

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

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



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)

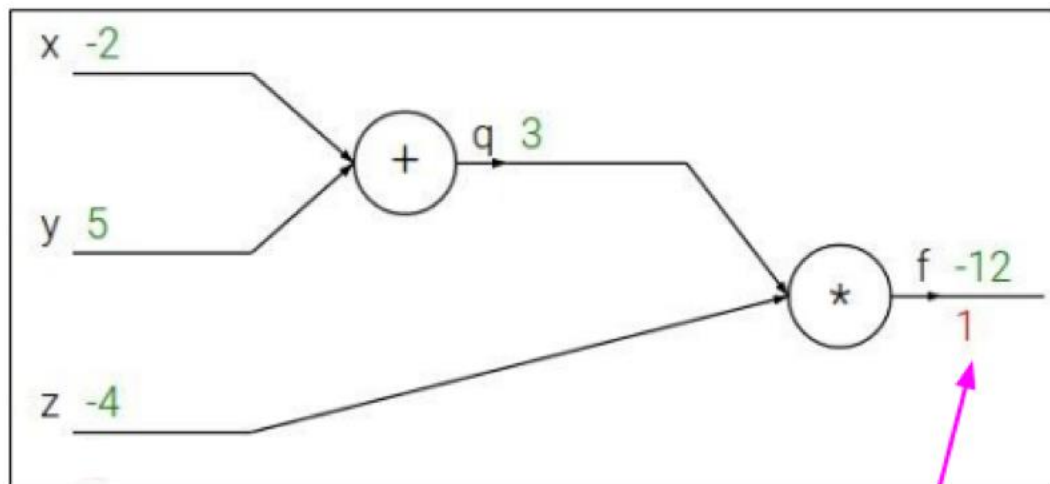
$$f(x, y, z) = (x + y)z$$

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

Node
Gradients

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

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



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

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)

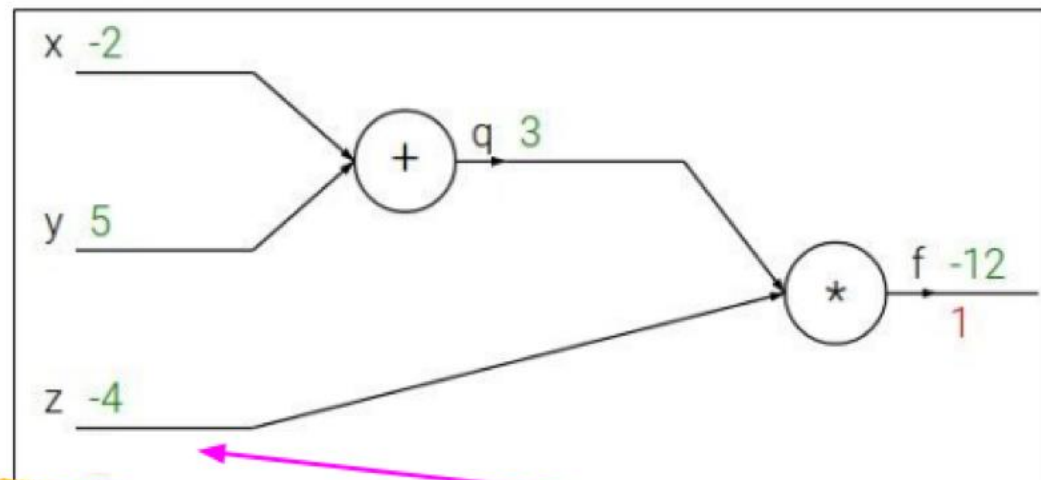
$$f(x, y, z) = (x + y)z$$

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

Node
Gradients

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

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



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

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)

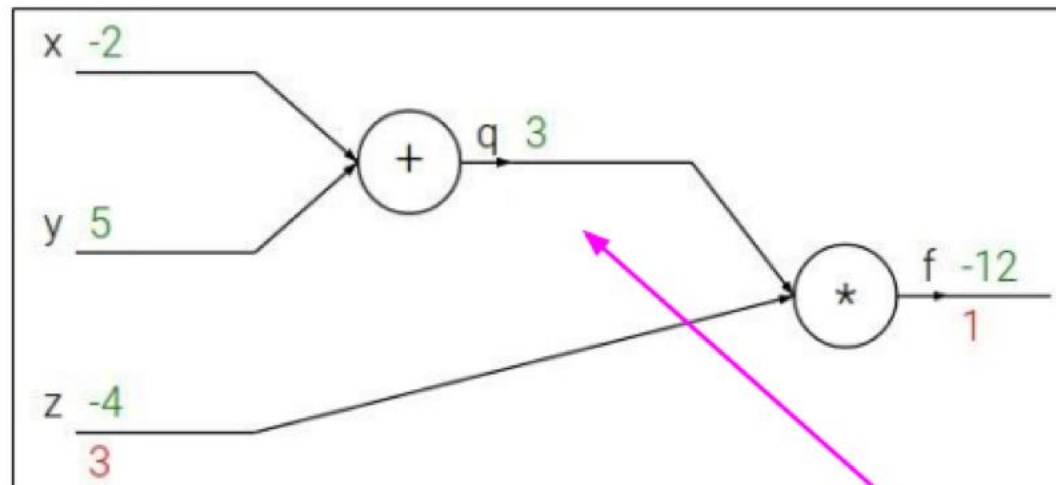
$$f(x, y, z) = (x + y)z$$

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

Node
Gradients

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

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



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

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



Example On Simple Function gradients to maximize f (at a point)

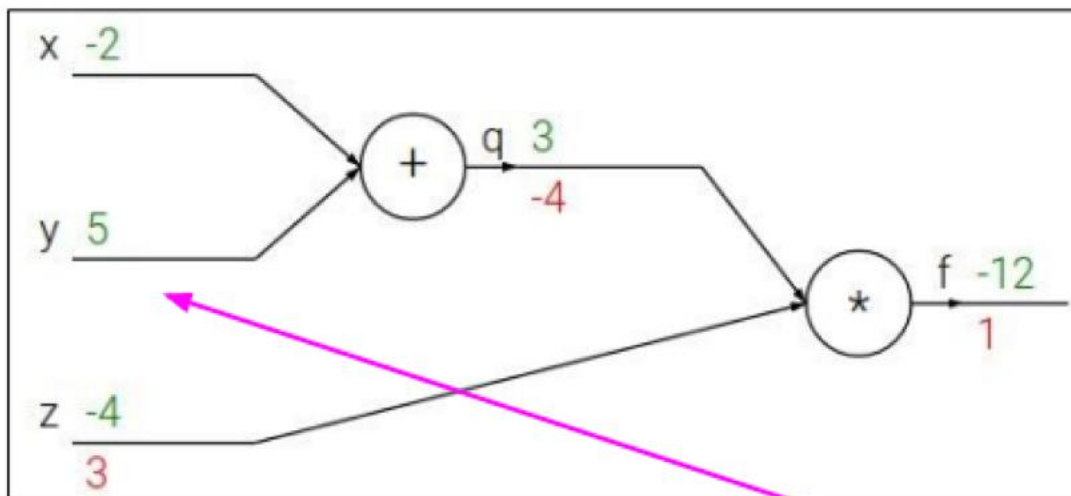
$$f(x, y, z) = (x + y)z$$

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

Node
Gradients

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

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



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

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)

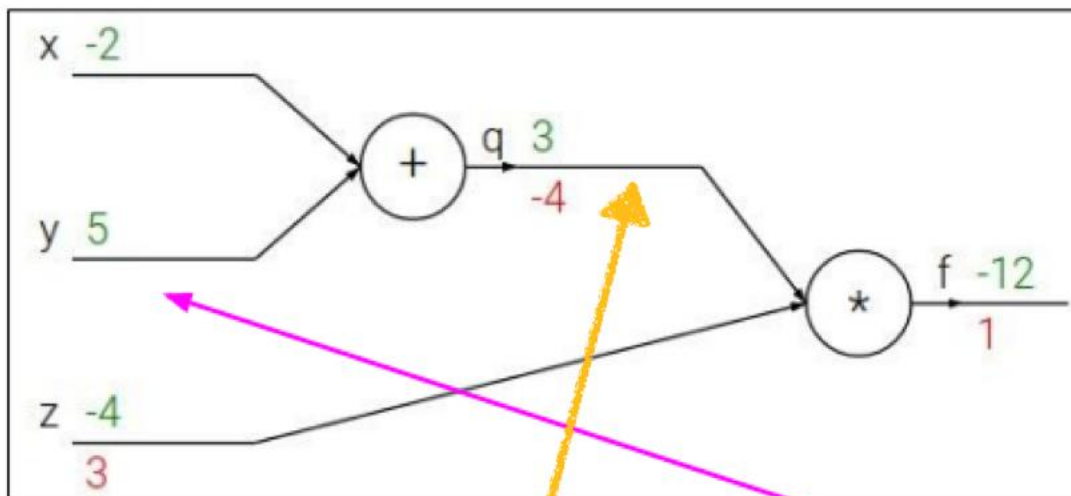
$$f(x, y, z) = (x + y)z$$

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

Node
Gradients

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

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



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

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)

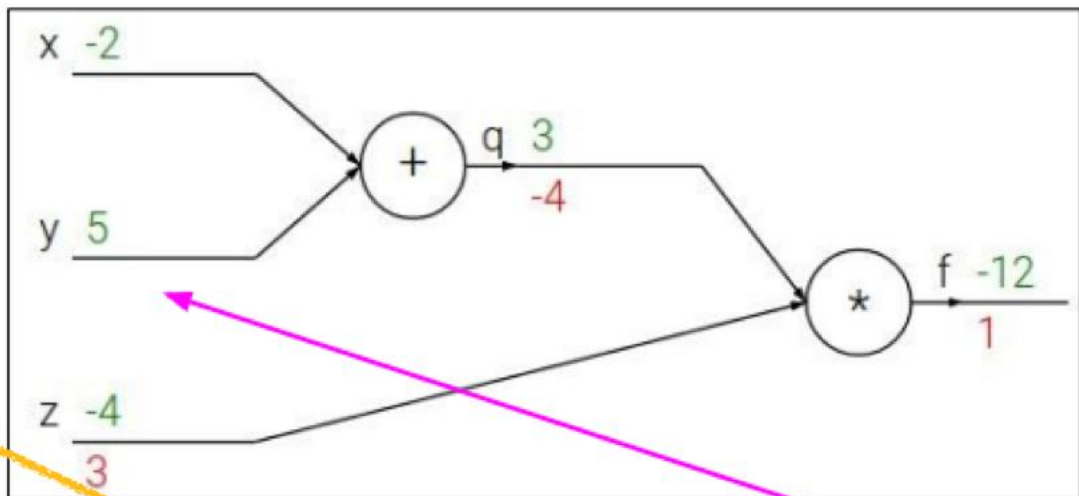
$$f(x, y, z) = (x + y)z$$

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

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

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

Node
Gradients



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

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)

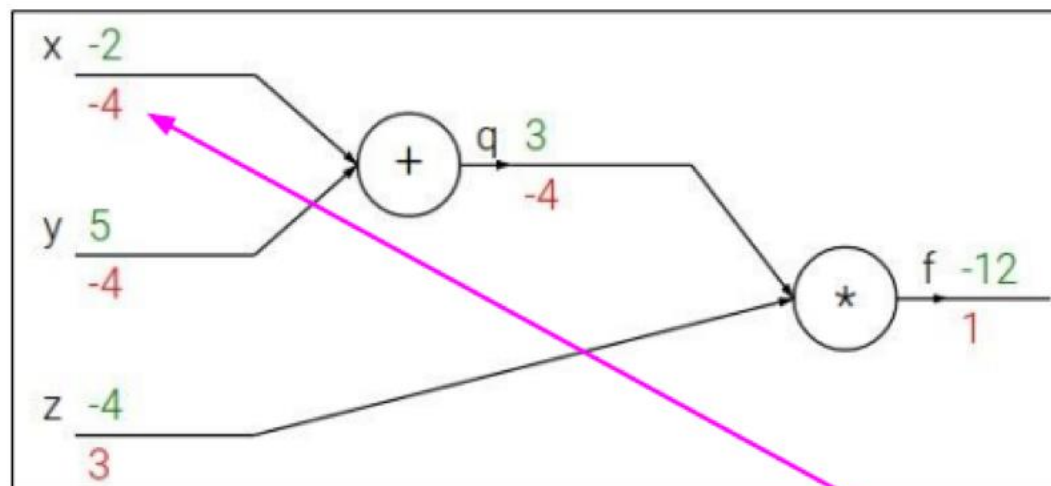
$$f(x, y, z) = (x + y)z$$

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

Node
Gradients

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

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



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

Chain rule:

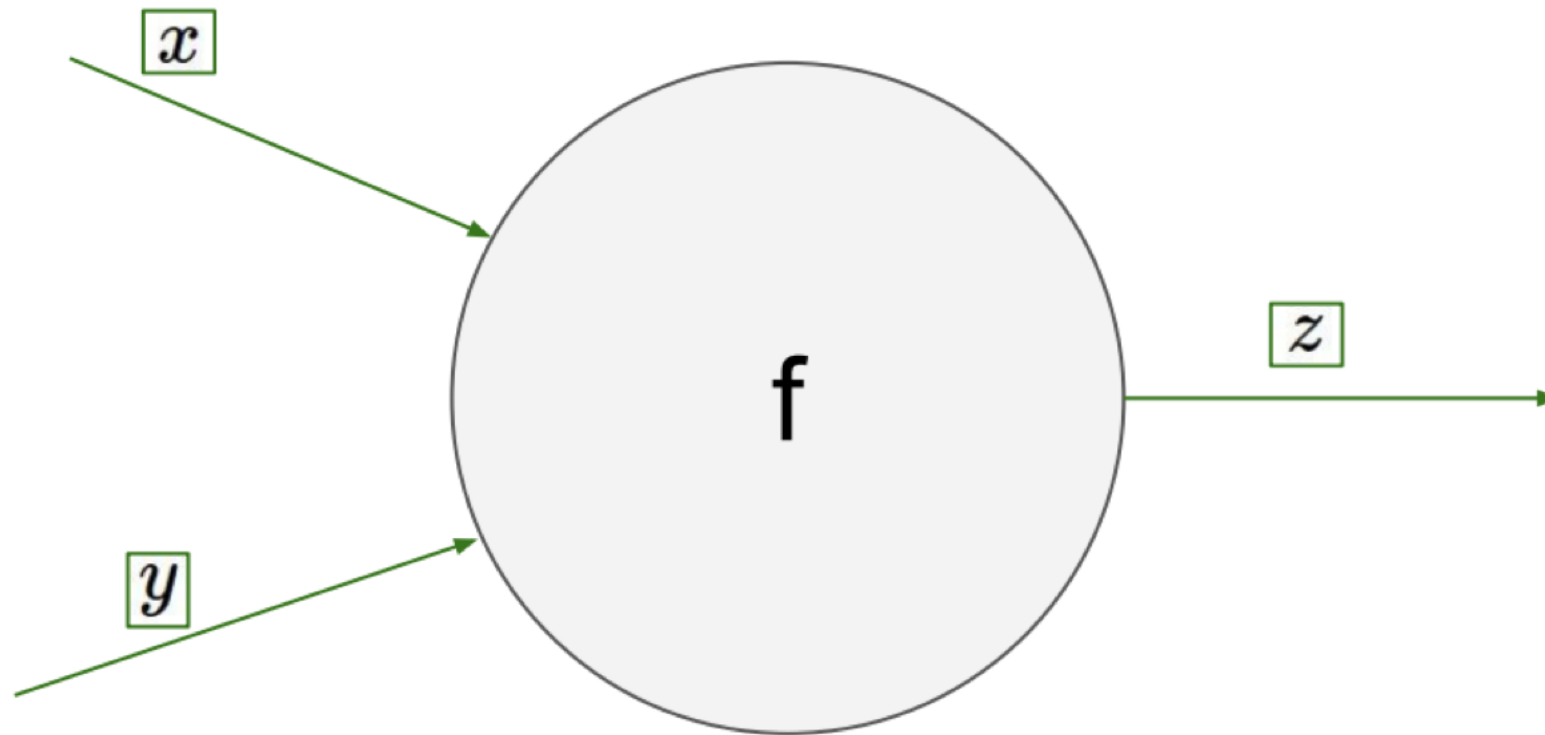
$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial x}$$

Upstream
gradient

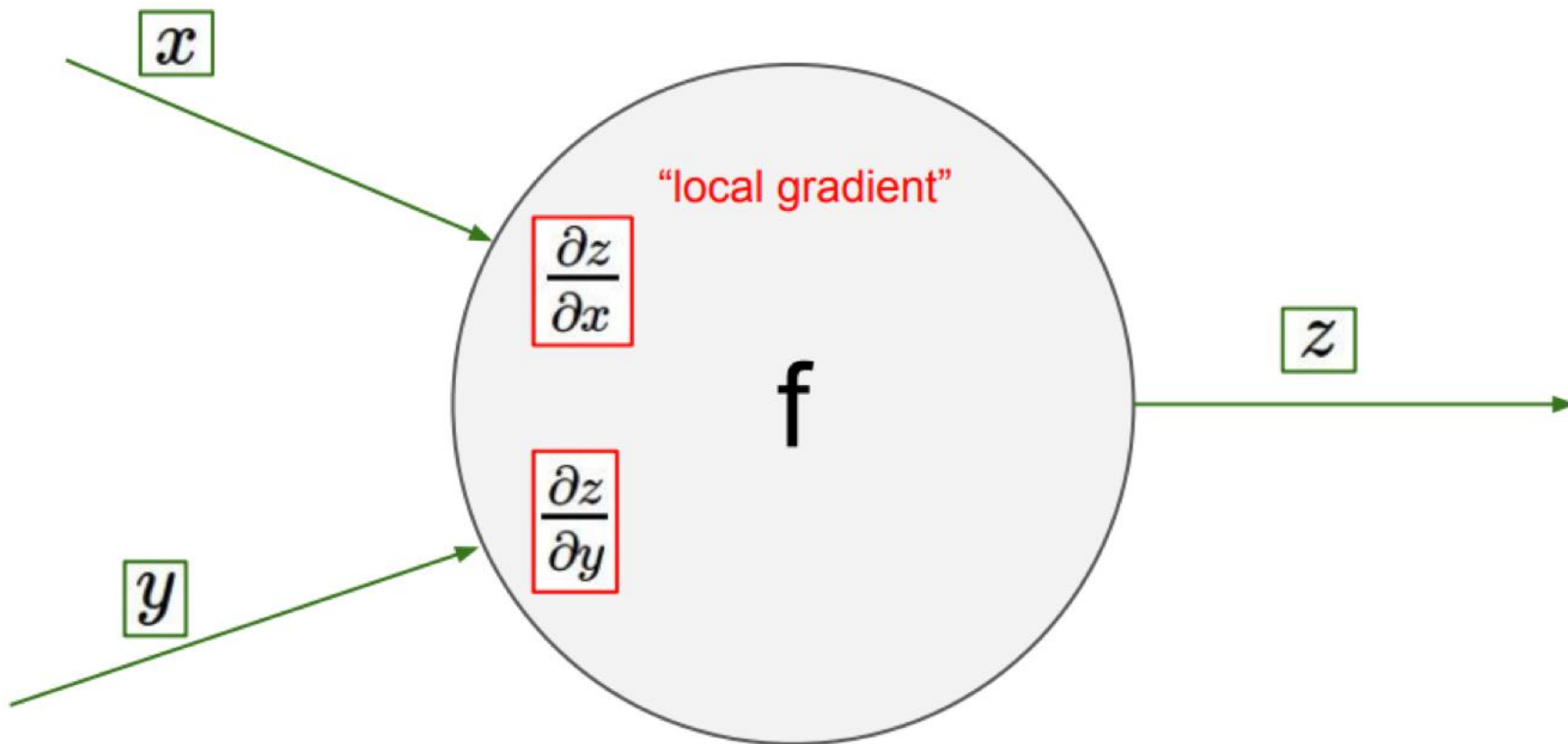
Local
gradient

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

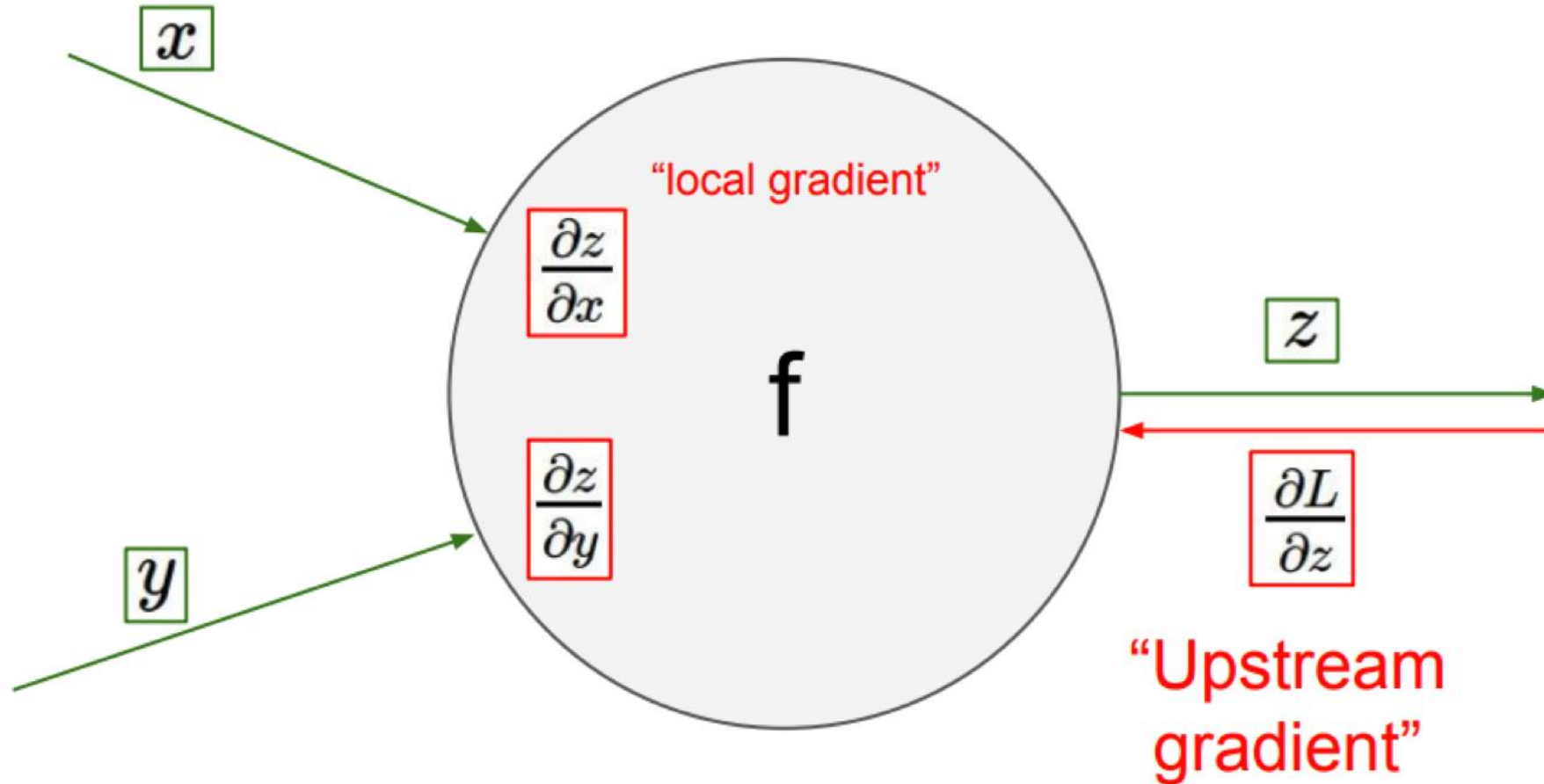
General Rule



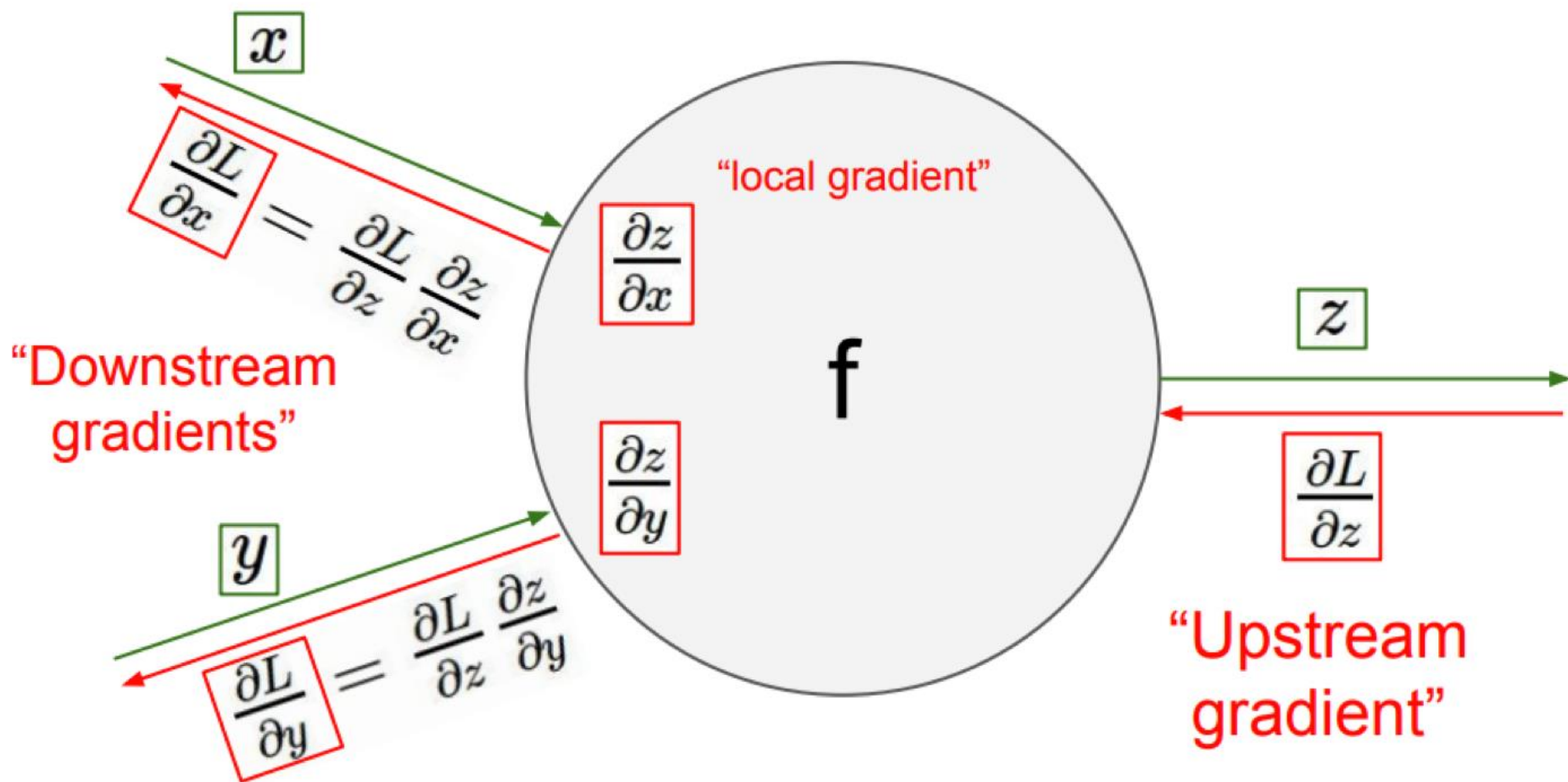
General Rule



General Rule



General Rule



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

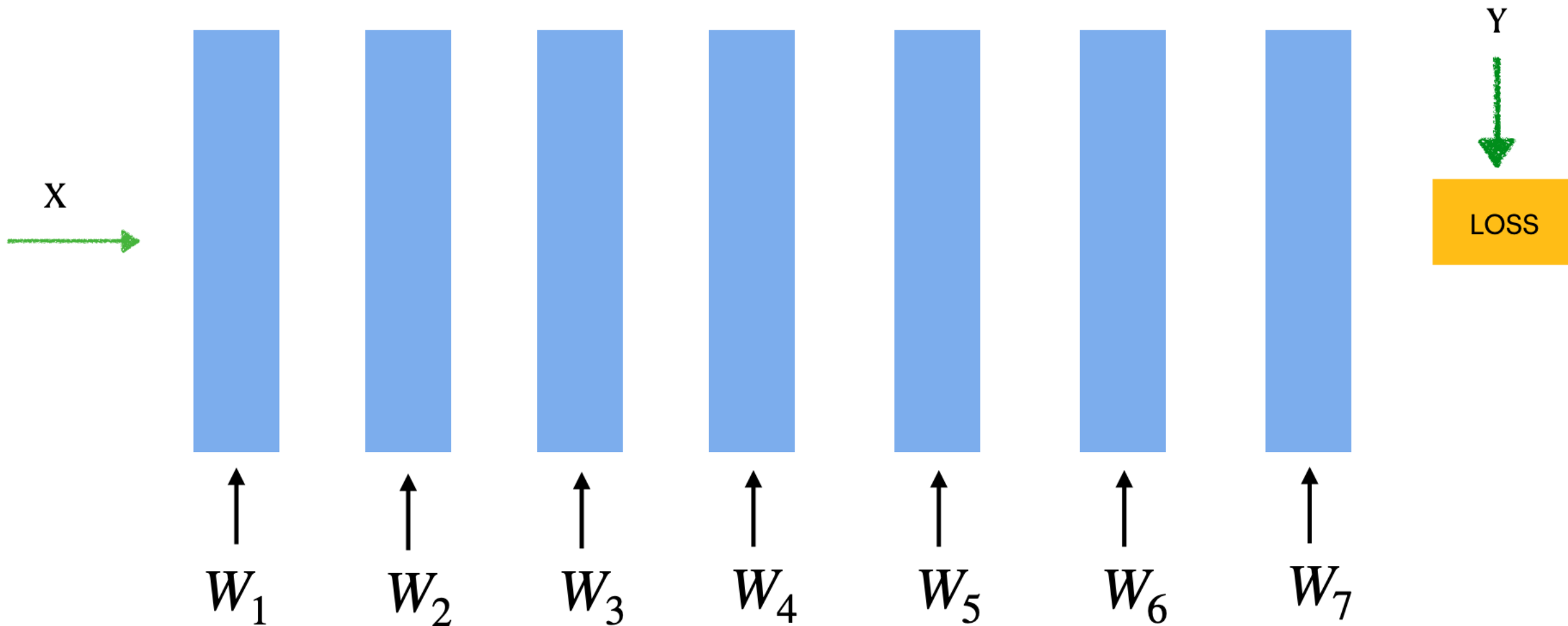
Backpropagation in general computational graph

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

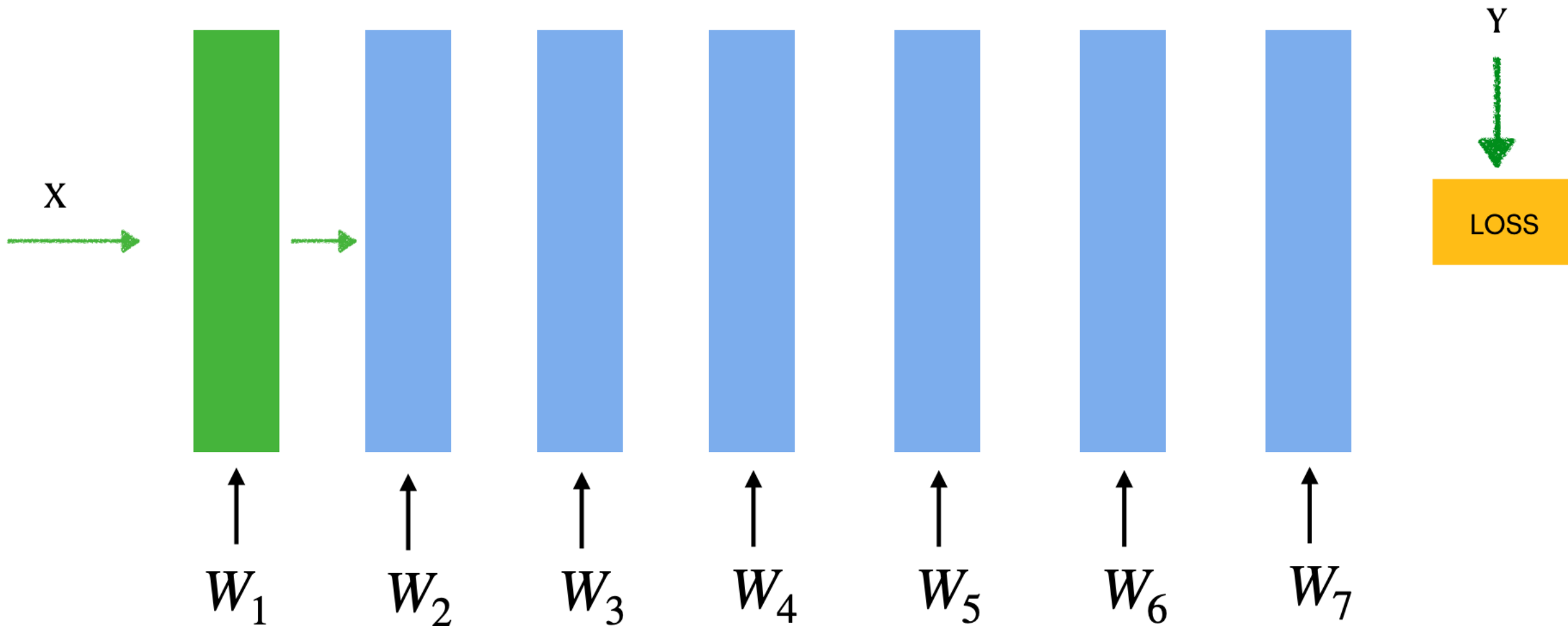
$$\frac{\partial L}{\partial x} = \sum_{i=1}^n \frac{\partial \mathcal{L}}{\partial y_i} \frac{\partial y_i}{\partial x}$$

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

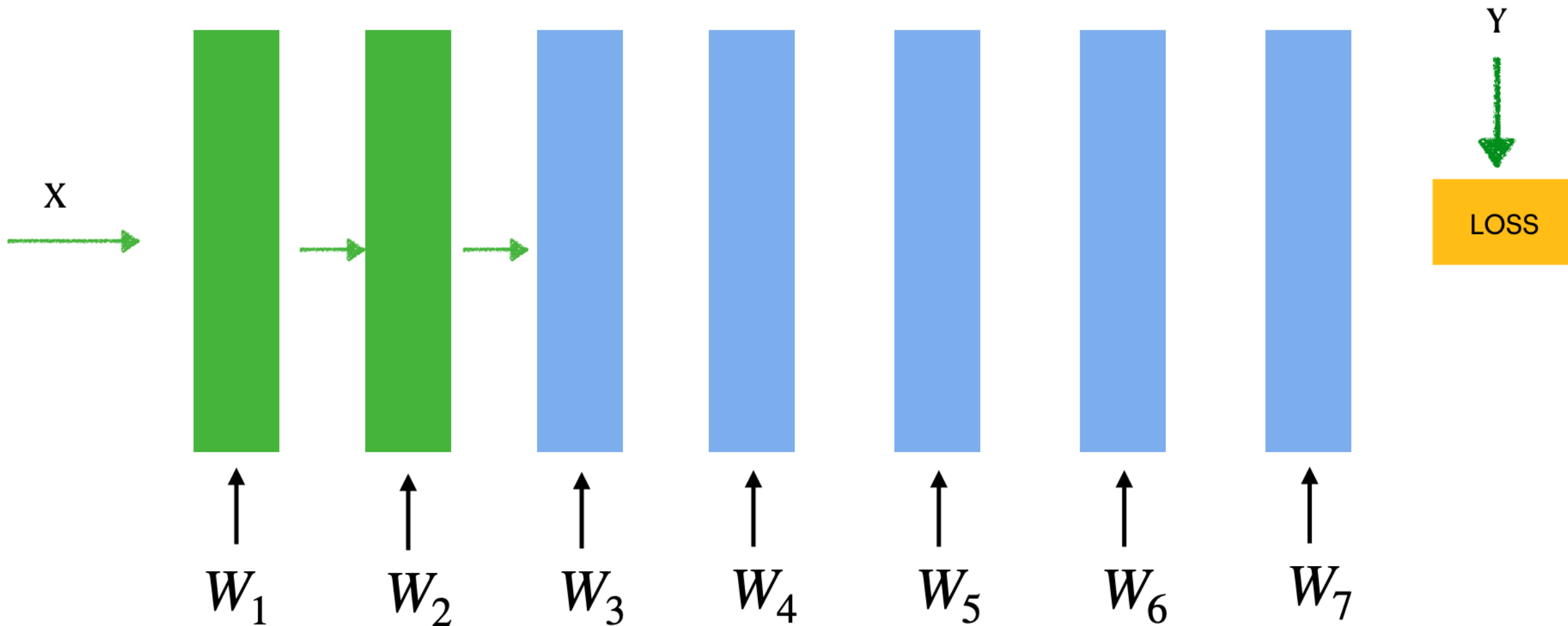
Forward Computation



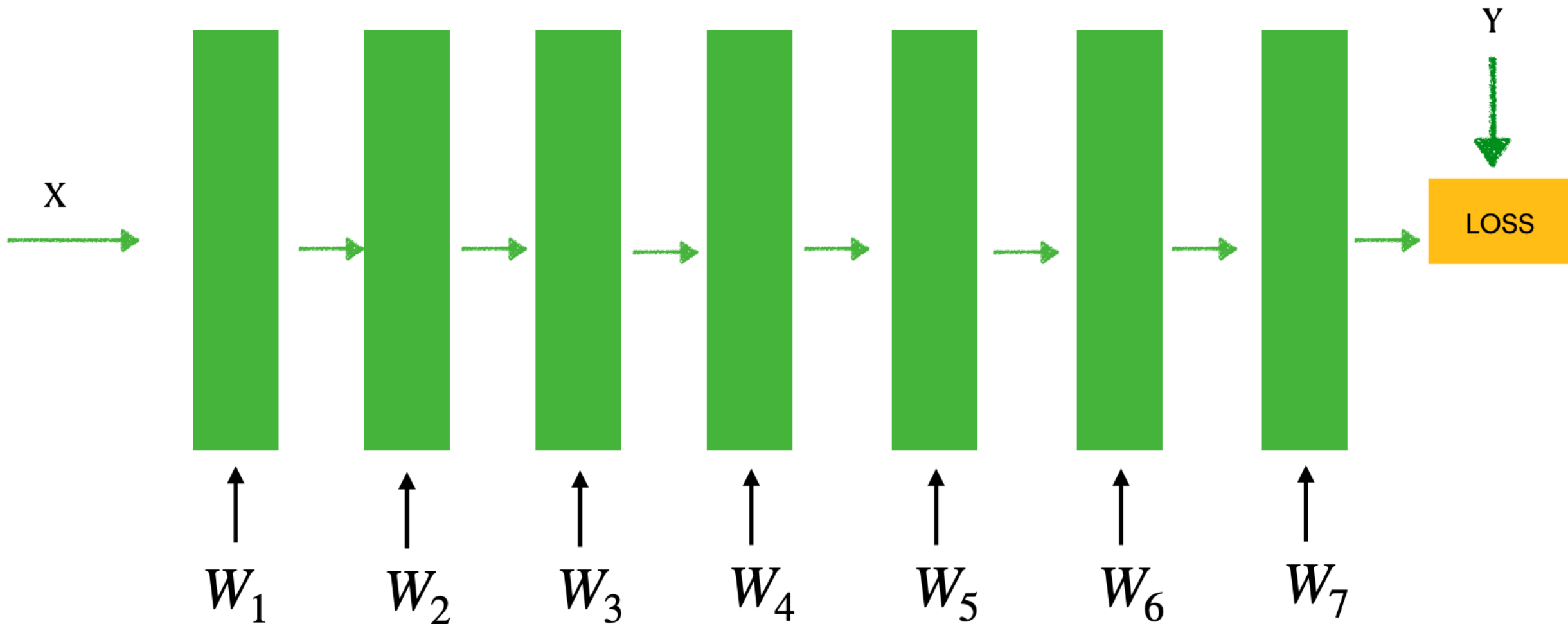
Forward Computation



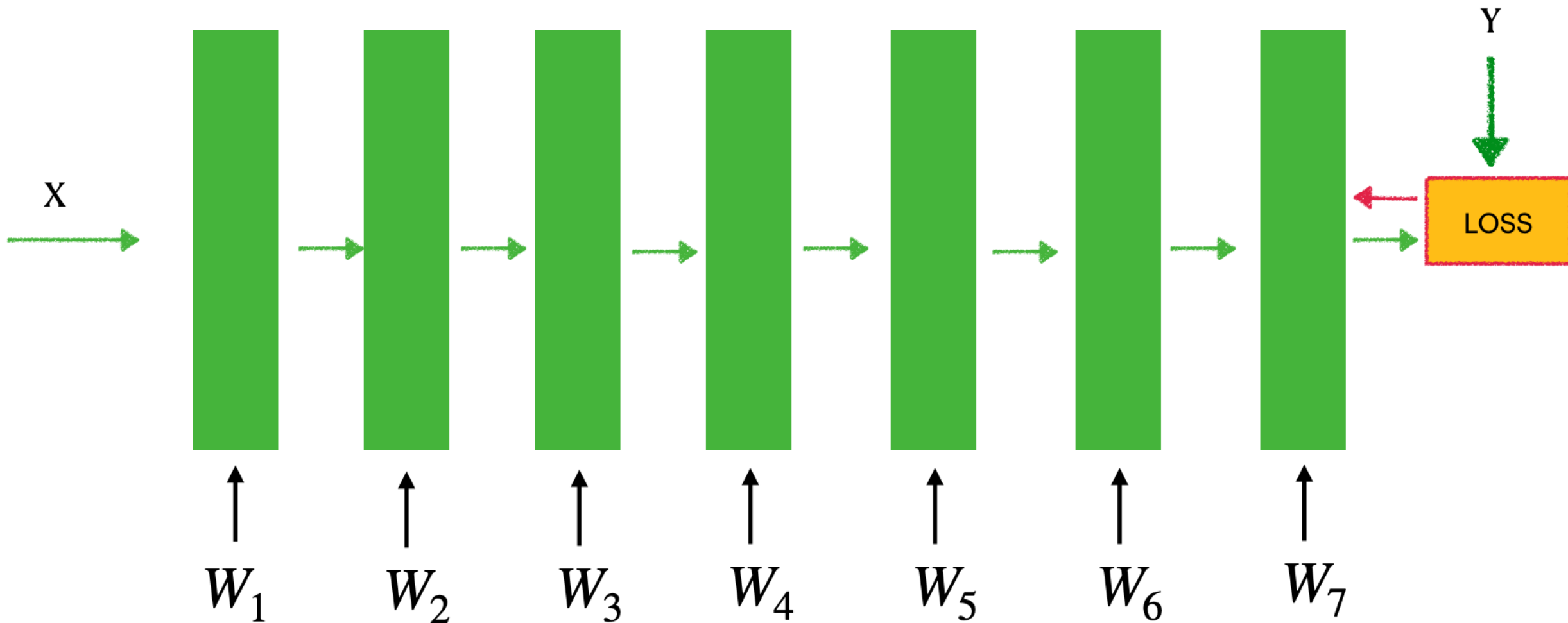
Forward Computation



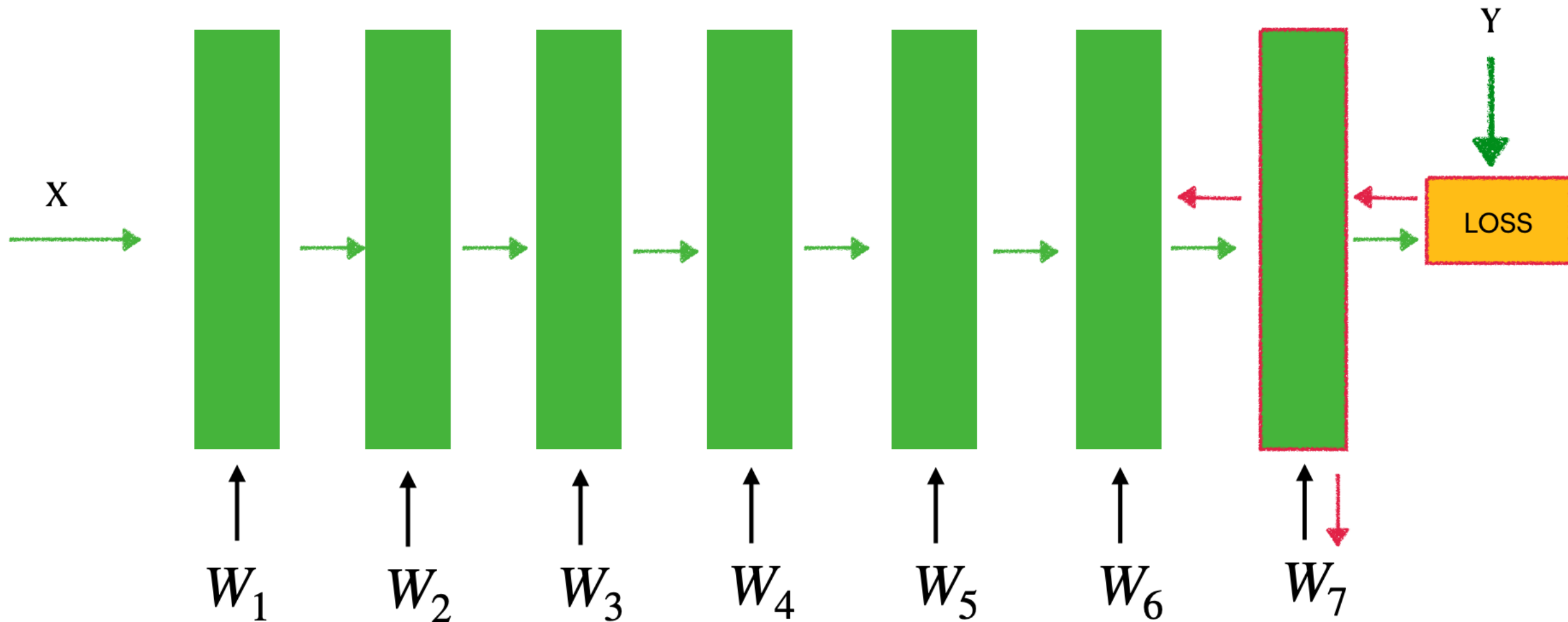
Forward Computation



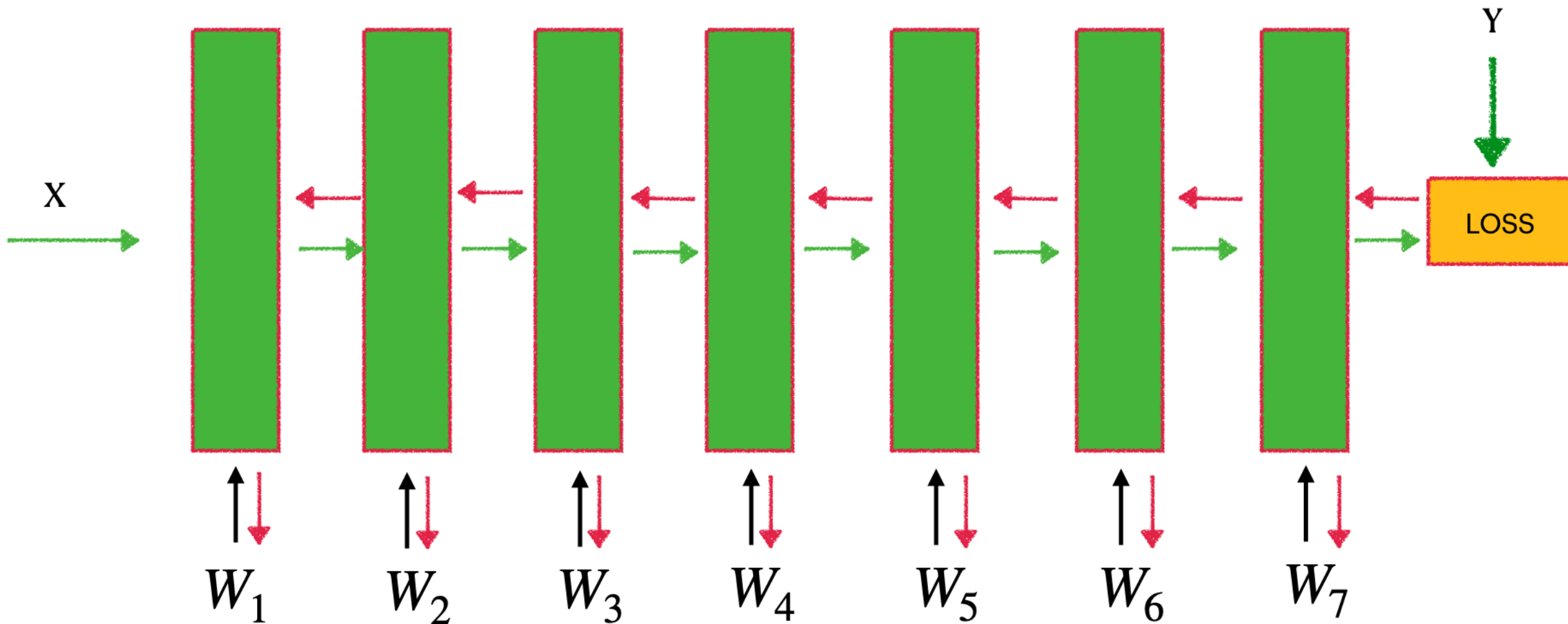
Backward Computation



Backward Computation



Backward Computation



Common Frameworks Do this Automatically

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



<https://www.tensorflow.org/>



<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

