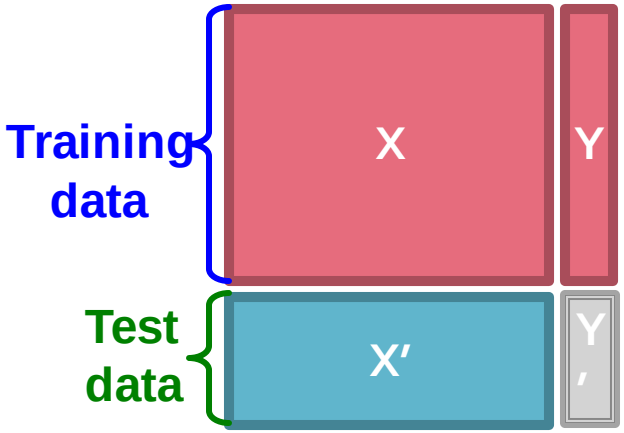
Large-Scale Machine Learning: Neural Nets

CS246: Mining Massive Datasets
Jure Leskovec, Stanford University
http://cs246.stanford.edu

Supervised Learning

- **Task:** Given data **(X,Y)** build a model **f** to predict **Y'** based on **X'**
- **Strategy:** **Estimate $y = f(x)$ on (X, Y).**
  Hope that the same $f(x)$ also works to predict unknown Y'

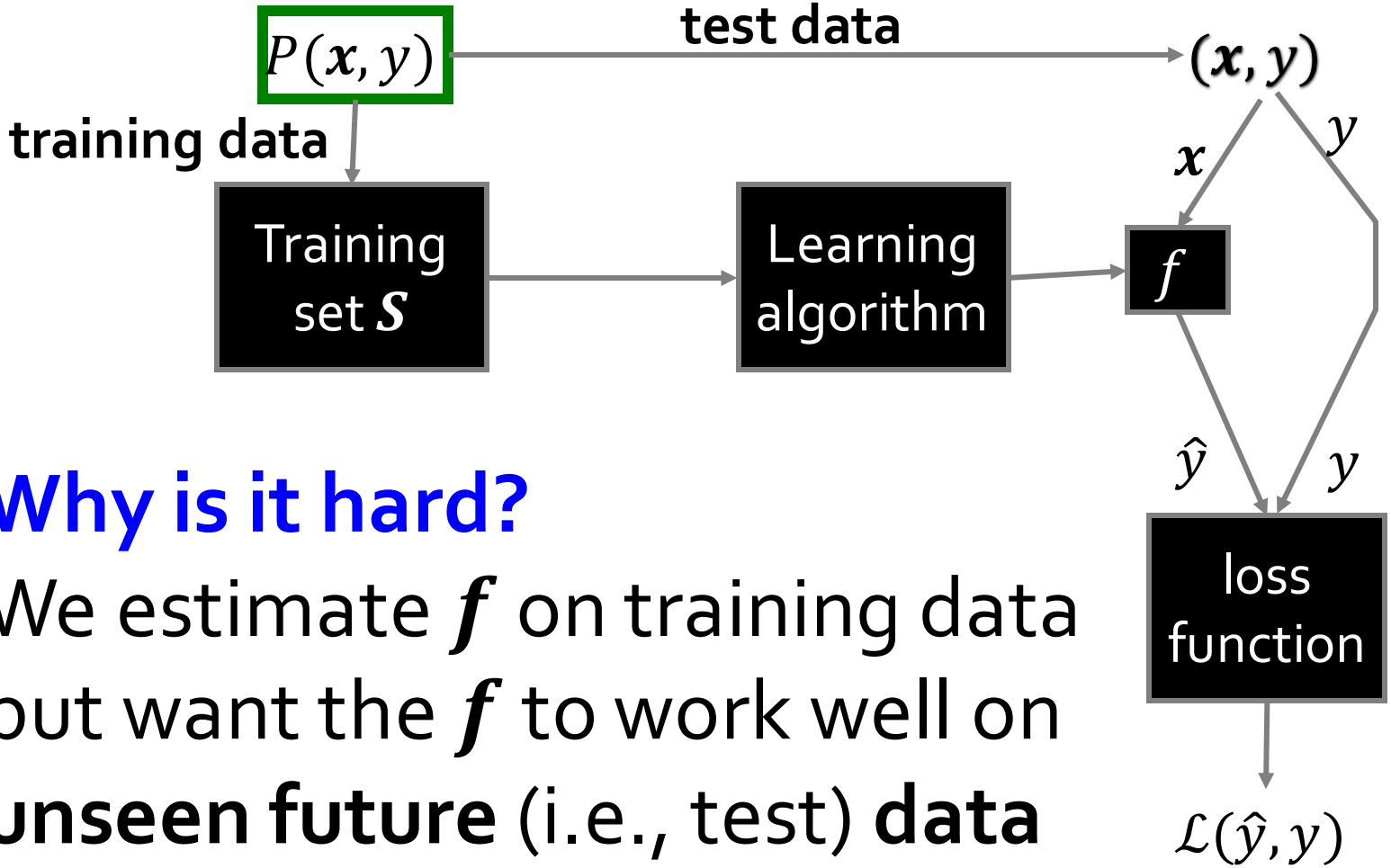


  - The **"hope"** is called **generalization**
    - **Overfitting:** **If f(x) predicts Y well but is unable to predict Y'**
  - **We want to build a model that generalizes well to unseen data**

Formal Setting

- **1)** Training data is drawn independently at random according to unknown probability distribution $P(\boldsymbol{x}, y)$
- **2)** The learning algorithm analyzes the examples and produces a classifier $\boldsymbol{f}$
- Given **new** data $(\boldsymbol{x}, y)$ drawn from $\boldsymbol{P}$, the classifier is given $\boldsymbol{x}$ and predicts $\hat{\boldsymbol{y}} = \boldsymbol{f}(\boldsymbol{x})$
- The **loss** $\mathcal{L}(\hat{\boldsymbol{y}}, \boldsymbol{y})$ is then measured
- **Goal of the learning algorithm:**
  Find $\boldsymbol{f}$ that minimizes **expected loss** $E_{\boldsymbol{P}}[\mathcal{L}]$

Formal Setting



Why is it hard?

We estimate f on training data but want the f to work well on **unseen future** (i.e., test) **data**

Minimizing the Loss

- **Goal: Minimize the expected loss**

$$\min_f \mathbb{E}_P[\mathcal{L}]$$

- **But we don't have access to P -- we only know the training data D:**

$$\min_f \mathbb{E}_D[\mathcal{L}]$$

- **So, we minimize the average loss on the training data:**

$$\min_f J(f) = \min_f \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(f(x_i), y_i)$$

Problem: Just memorizing the training data gives us a perfect model (with zero loss)

ML == Optimization

- **Given:**
  - A set of **N** training examples
    - $\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$
  - A loss function $\mathcal{L}$
- **Choose the model:** $f_\theta(x)$
- **Find:**
  - The parameter θ that minimizes the **expected loss on the training data**

$$\min_f J(f) = \min_f \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(f_\theta(x_i), y_i)$$

The Key Questions of ML

- What is the model $f_\theta(x)$?
- What is the loss $\mathcal{L}$?
- How do we optimize the loss?

What is the model $f_\theta(x)$?

- θ refers to **all the parameters of a model** that we optimize
- Examples:
  - A "Linear layer" $f_\theta(x) = Wx + b$
    - Here $\theta = \{W, b\}$
  - A Multi-layer Perceptron (MLP)
    $$f_\theta(x) = W_2\sigma(W_1x + b_1) + b_2$$
    - Here $\theta = \{W_1, W_2, b_1, b_2\}$
    - σ is an activation function (we will come to this later)

What is the loss $\mathcal{L}$?

- $\mathcal{L}$: **loss function**. Example: L2 loss

$$\mathcal{L}\big(\boldsymbol{y}, f(\boldsymbol{x})\big) = \|y - f(x)\|_2$$

  - Common loss functions for **regression**:
    - L2 loss, L1 loss, huber loss, …
  - Common loss functions for **classification**:
    - Cross entropy, max margin (hinge loss), …
  - Example
    - See https://pytorch.org/docs/stable/nn.html#loss-functions

Loss Function Example

- One common loss for classification: **cross entropy (CE)**
- Label $\boldsymbol{y}$ is a categorical vector (one-hot encoding)
  - e.g. $\boldsymbol{y} =$ | 0 | 0 | 1 | 0 | 0 |

y is of class "3"

- $f(\boldsymbol{x}) = \text{Softmax}(g(\boldsymbol{x}))$
  - $f(\boldsymbol{x})_i = \frac{e^{g(x)_i}}{\sum_{j=1}^{C} e^{g(x)_j}}$

C is the number of classes (5 here)

$g(x)_i$ denotes i-th coordinate of the vector output of func. $g(x)$

  - Softmax normalizes a vector into a probability distribution that sums to 1
  - e.g. $f(\boldsymbol{x}) =$ | 0.1 | 0.3 | 0.4 | 0.1 | 0.1 |

- $\text{CE}(\boldsymbol{y}, f(\boldsymbol{x})) = -\sum_{i=1}^{C}(y_i \log f(x)_i)$

$$= -\log(f(x)_{correct\_class})$$

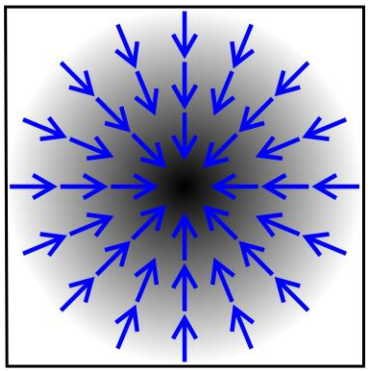
  - $y_i, f(x)_i$ are the **actual** and **predicted** value of the i-th class.
- **Intuition:** the lower the loss, the closer the prediction is to one-hot $\boldsymbol{y}$

Machine Learning as Optimization

- **How to optimize the objective function?**
- **Gradient vector:** Direction and rate of fastest increase

Partial derivative

$$\nabla_{\Theta}\mathcal{L} = \left(\frac{\partial \mathcal{L}}{\partial \Theta_1}, \frac{\partial \mathcal{L}}{\partial \Theta_2}, \ldots\right)$$



https://en.wikipedia.org/wiki/Gradient

  - Θ_1, Θ_2 … : components of Θ
- Recall **directional derivative** of a multi-variable function (e.g. $\mathcal{L}$) along a given vector represents the instantaneous rate of change of the function along the vector.
- Gradient is the directional derivative in the **direction of largest increase**

Gradient Descent

- **Iterative algorithm:** repeatedly update weights in the (opposite) direction of gradients until convergence

$$\Theta \leftarrow \Theta - \eta \nabla_{\Theta} \mathcal{L}$$

- **Training:** Optimize Θ iteratively
  - **Iteration**: 1 step of gradient descent
- **Learning rate (LR) η:**
  - Hyperparameter that controls the size of gradient step
  - Can vary over the course of training (LR scheduling)
- **Ideal termination condition: 0** gradient
  - In practice, we stop training if it no longer improves performance on the **validation set** (part of dataset we hold out from training)

Stochastic Gradient Descent (SGD)

- **Problem with gradient descent:**
  - Exact gradient requires computing $\nabla_\Theta \mathcal{L}(\boldsymbol{y}, f(\boldsymbol{x}))$, where $\boldsymbol{x}$ is the **entire** dataset!
    - This means summing gradient contributions over all the points in the dataset
    - Modern datasets often contain billions of data points
    - Extremely expensive for every gradient descent step
- **Solution: Stochastic gradient descent (SGD)**
  - At every step, pick a different **minibatch** $\boldsymbol{\mathcal{B}}$ containing a subset of the dataset, use it as input $\boldsymbol{x}$

Minibatch SGD

- **Concepts:**
  - **Batch size**: the number of data points in a minibatch
    - E.g. number of nodes for node classification task
  - **Iteration**: 1 step of SGD on a minibatch
  - **Epoch**: one full pass over the dataset (# iterations is equal to ratio of dataset size and batch size)

- **SGD is unbiased estimator of full gradient:**
  - But there is no guarantee on the rate of convergence
  - In practice often requires tuning of learning rate
- Common optimizer that improves over SGD:
  - Adam, Adagrad, Adadelta, RMSprop …

Neural Network Function

- **Objective:** $\min_{\Theta} \mathcal{L}(\boldsymbol{y}, f(\boldsymbol{x}))$
- In deep learning, the function f can be very complex
- To start simple, consider linear function

$$f(\boldsymbol{x}) = W \cdot \boldsymbol{x}, \qquad \Theta = \{W\}$$

- If f returns a scalar, then W is a learnable **vector**

$$\nabla_W f = \left(\frac{\partial f}{\partial w_1}, \frac{\partial f}{\partial w_2}, \frac{\partial f}{\partial w_3} \ldots\right)$$

- If f returns a vector, then W is the **weight matrix**

$$\nabla_W f = W^T$$

Jacobian matrix of f

Back-propagation

- **How about a more complex function:**

$$f(\boldsymbol{x}) = W_2(W_1\boldsymbol{x}), \qquad \Theta = \{W_1, W_2\}$$

- Recall **chain rule**:

$$\frac{\mathrm{d}z}{\mathrm{d}x} = \frac{\mathrm{d}z}{\mathrm{d}y} \cdot \frac{\mathrm{d}y}{\mathrm{d}x}$$

In other words:

$f(\boldsymbol{x}) = W_2(W_1\boldsymbol{x})$

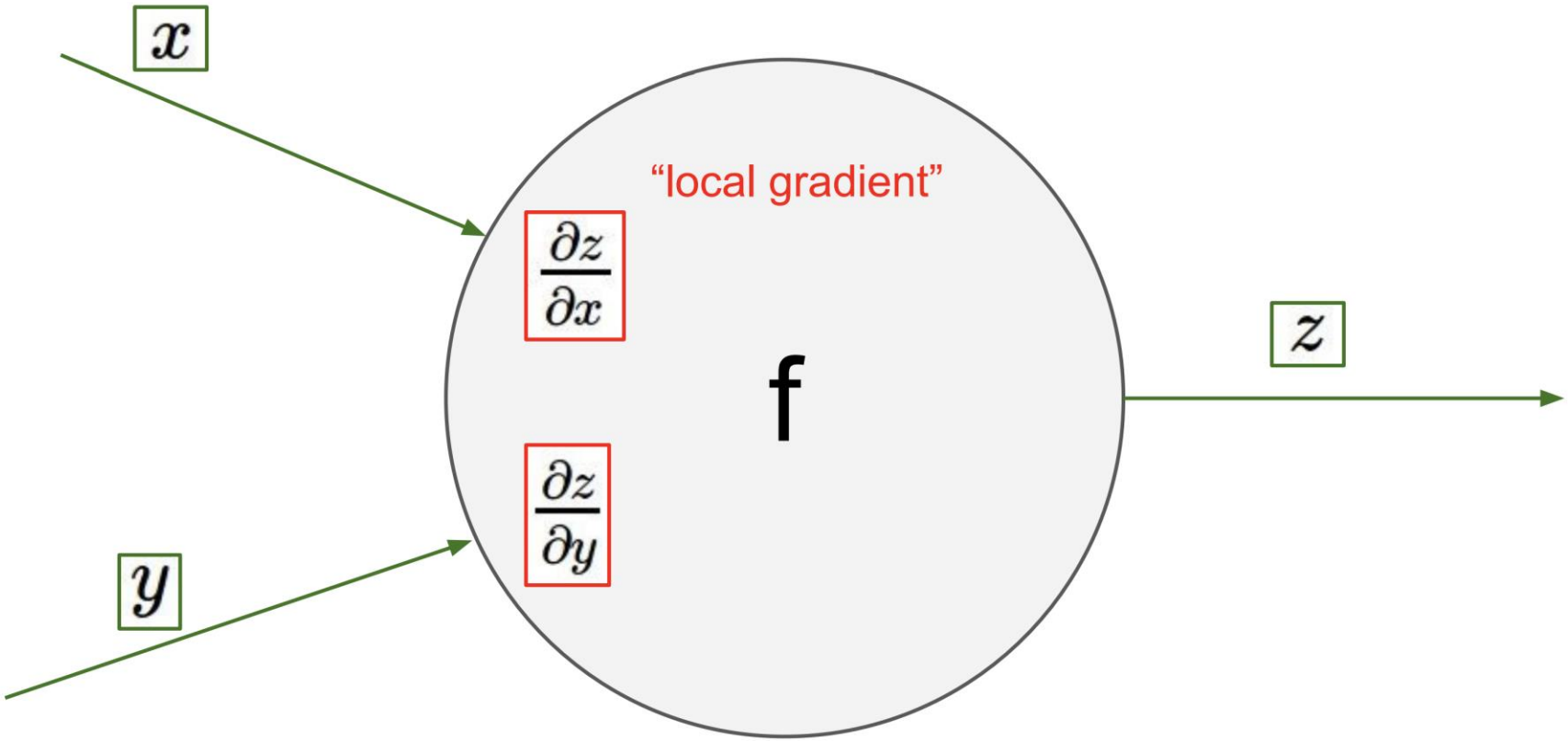
$h(x) = W_1\boldsymbol{x}$

$\mathrm{g}(z) = W_2 z$

- E.g. $\nabla_{\boldsymbol{x}} f = \frac{\partial f}{\partial (W_1\boldsymbol{x})} \cdot \frac{\partial (W_1\boldsymbol{x})}{\partial \boldsymbol{x}}$

- **Back-propagation**: Use of **chain rule** to propagate gradients of intermediate steps, and finally obtain gradient of $\mathcal{L}$ w.r.t. Θ

Back-propagation: General Setting



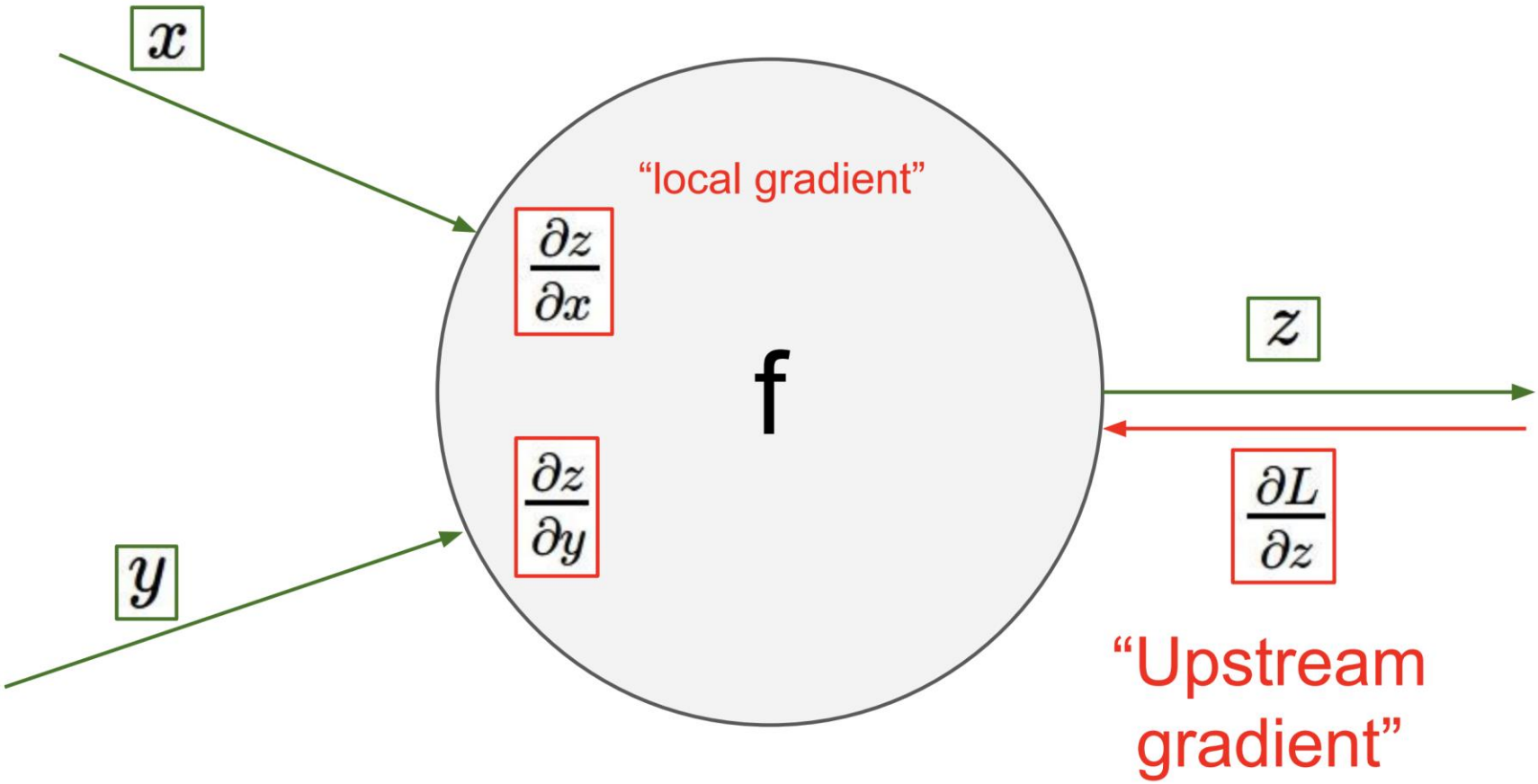
$$z = f(x, y)$$

Suppose we can write out the local gradients:

$$\frac{\partial z}{\partial x} \quad \frac{\partial z}{\partial y}$$

Credit: Stanford CS231n

Back-propagation: General Setting

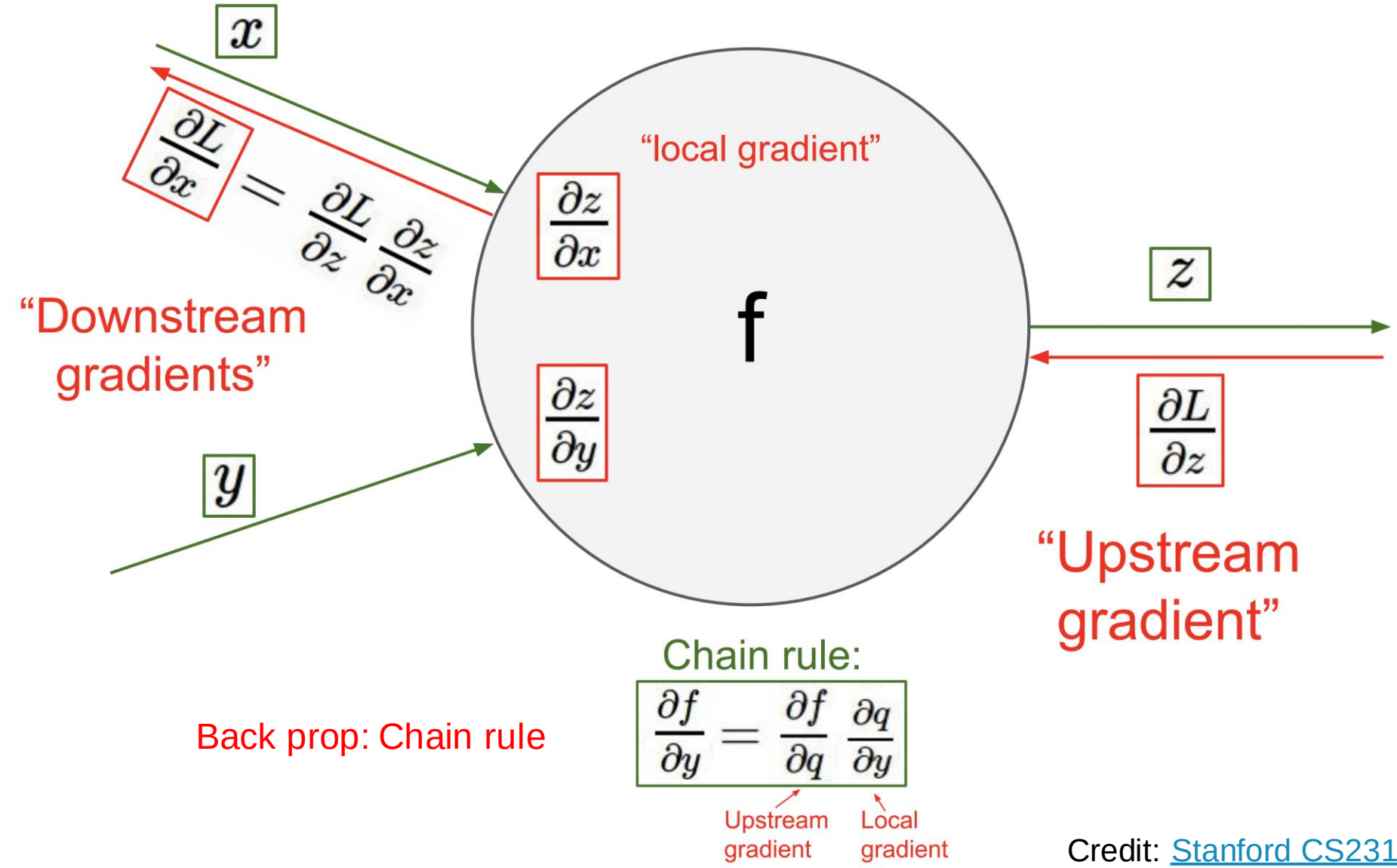


And we further have $L = g(z)$, with the upstream gradient $\frac{\partial L}{\partial z}$

How do we compute $\frac{\partial L}{\partial x}$ and $\frac{\partial L}{\partial y}$? Back prop!

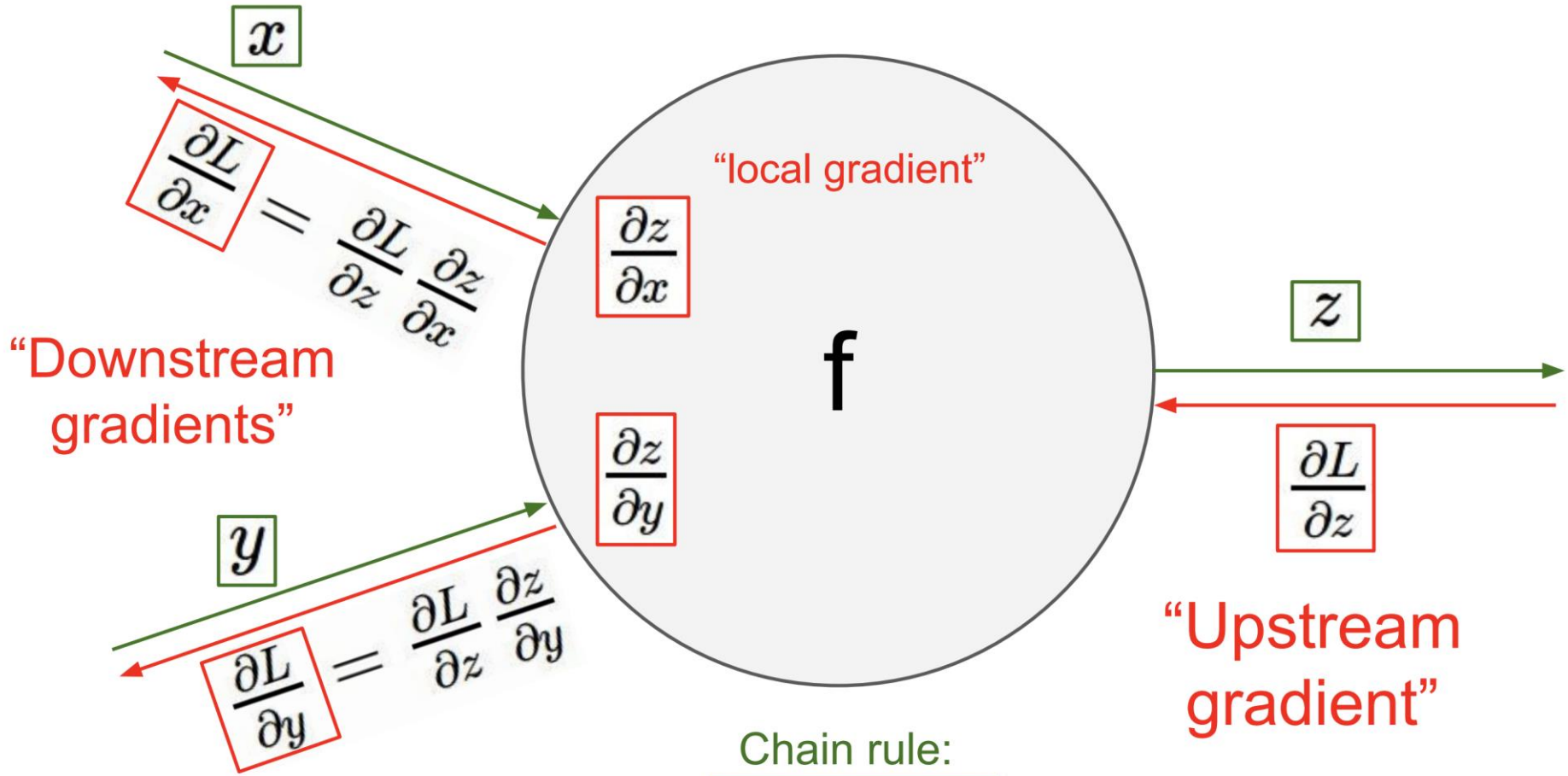
Credit: Stanford CS231n

Back-propagation: General Setting



Credit: Stanford CS231n

Back-propagation: General Setting



Back prop: Chain rule

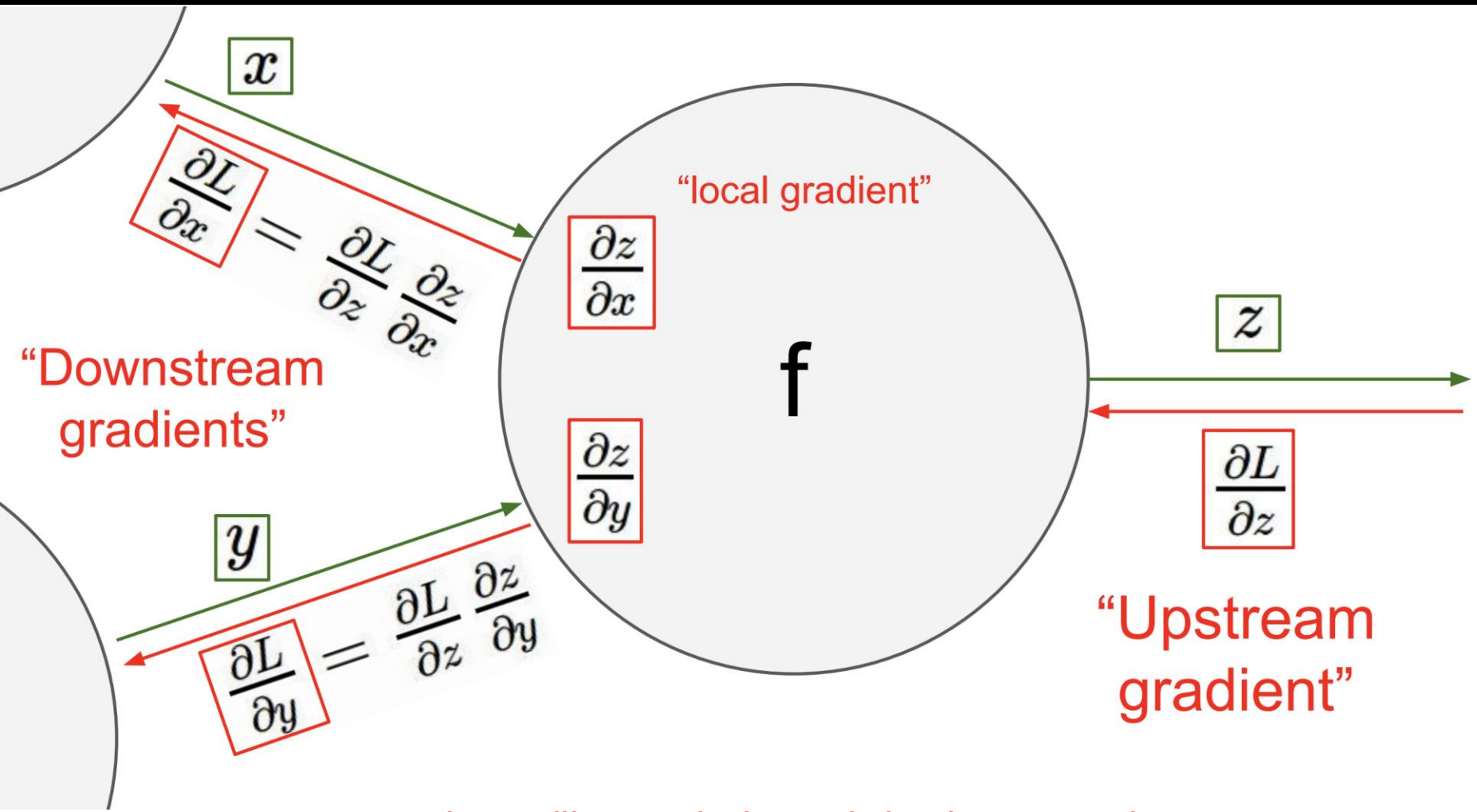
Chain rule:

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

Upstream gradient — Local gradient

Credit: Stanford CS231n

Back-propagation Example



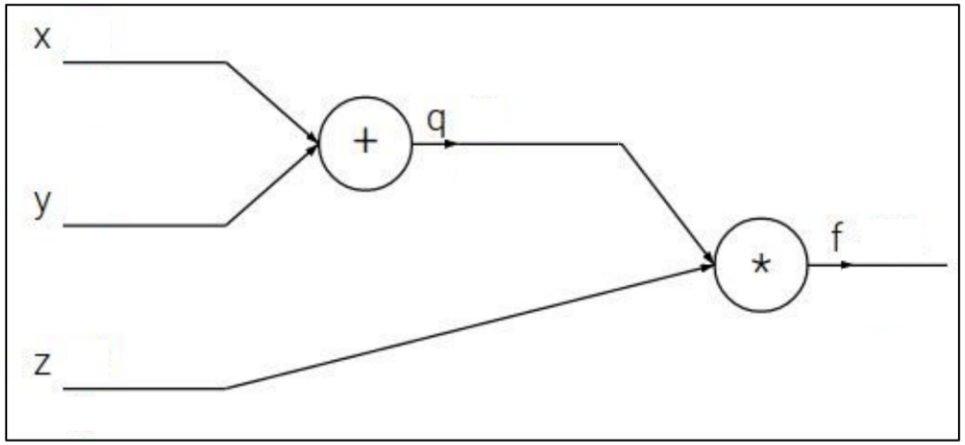
And we will recursively apply back propagation throughout the computation graph

Credit: Stanford CS231n

Back-propagation Example

Backpropagation: a simple example

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



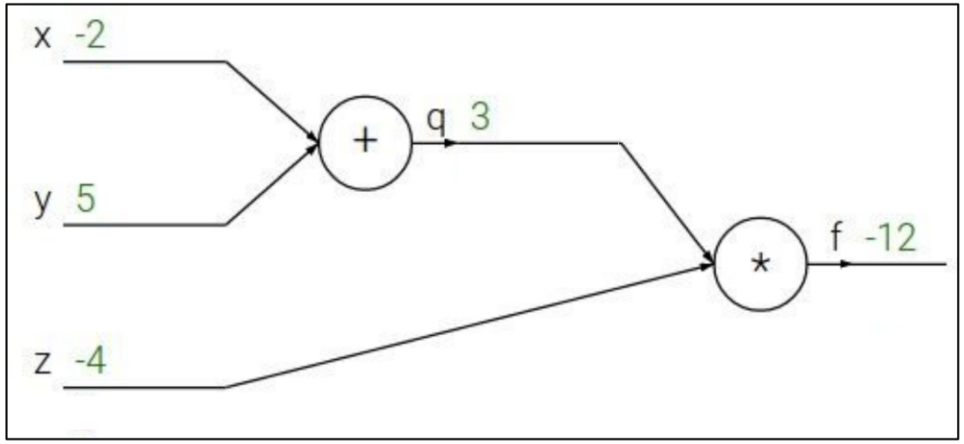
Credit: Stanford CS231n

Back-propagation Example

Backpropagation: a simple example

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

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



Credit: Stanford CS231n

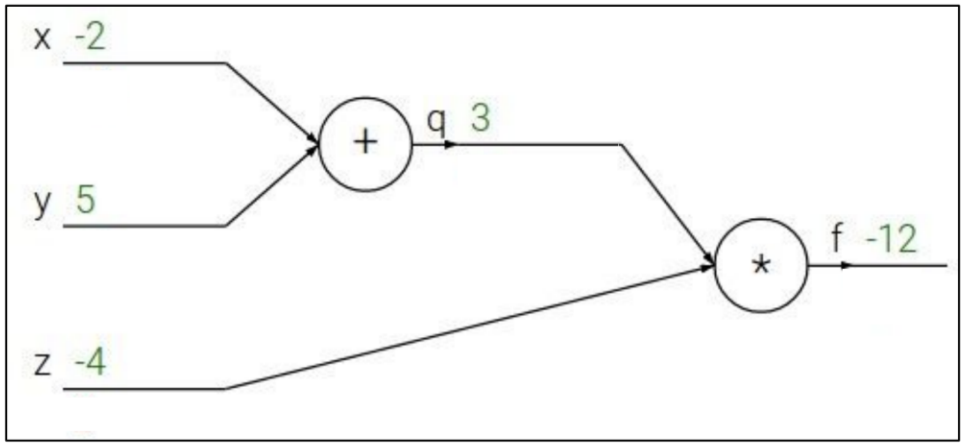
Back-propagation Example

Backpropagation: a simple example

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

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

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



Gradient computation for function q

Credit: Stanford CS231n

Back-propagation Example

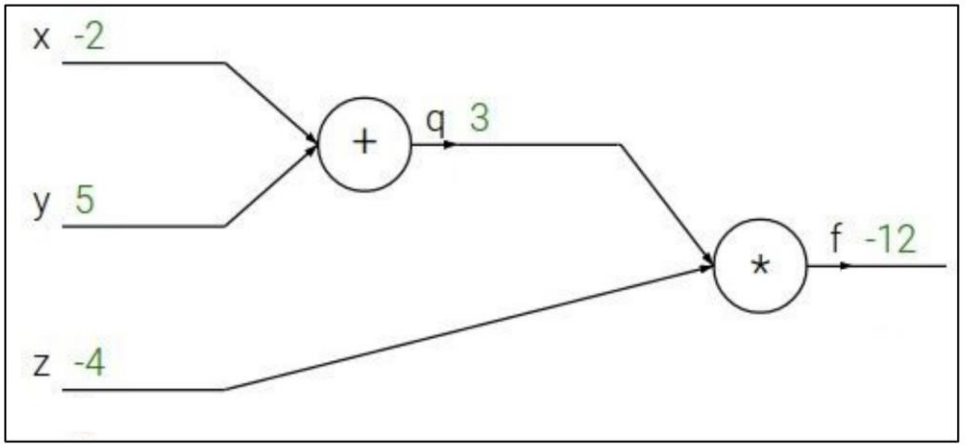
Backpropagation: a simple example

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

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

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

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



Gradient computation for function q

Gradient computation for function f

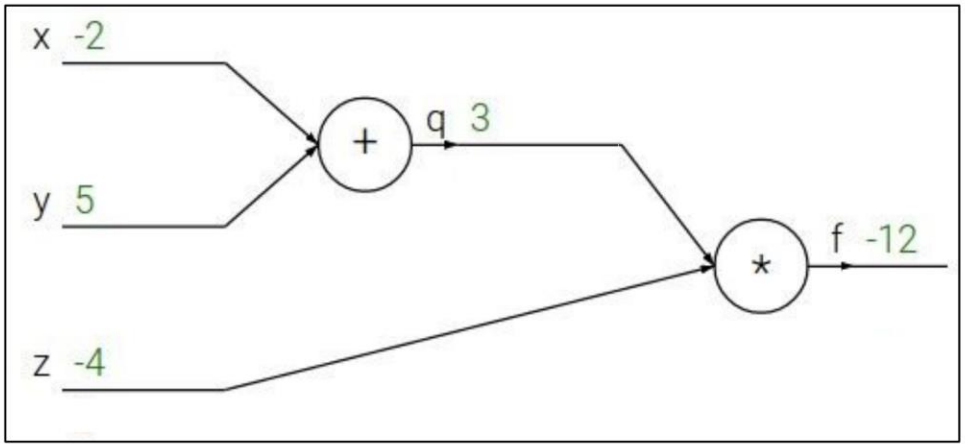
Credit: Stanford CS231n

Back-propagation Example

Backpropagation: a simple example

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

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



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

Gradient computation for function q

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

Gradient computation for function f

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

We want to compute gradients with respect to the input x, y, z

Credit: Stanford CS231n

Back-propagation Example

Backpropagation: a simple example

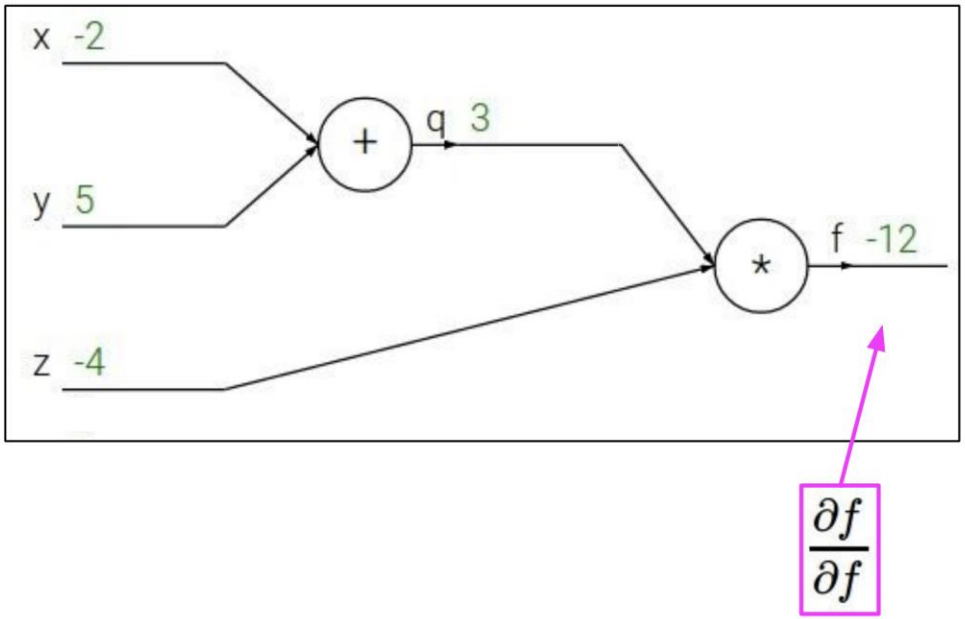
$$f(x, y, z) = (x + y)z$$

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



Let's start back prop!

Credit: Stanford CS231n

Back-propagation Example

Backpropagation: a simple example

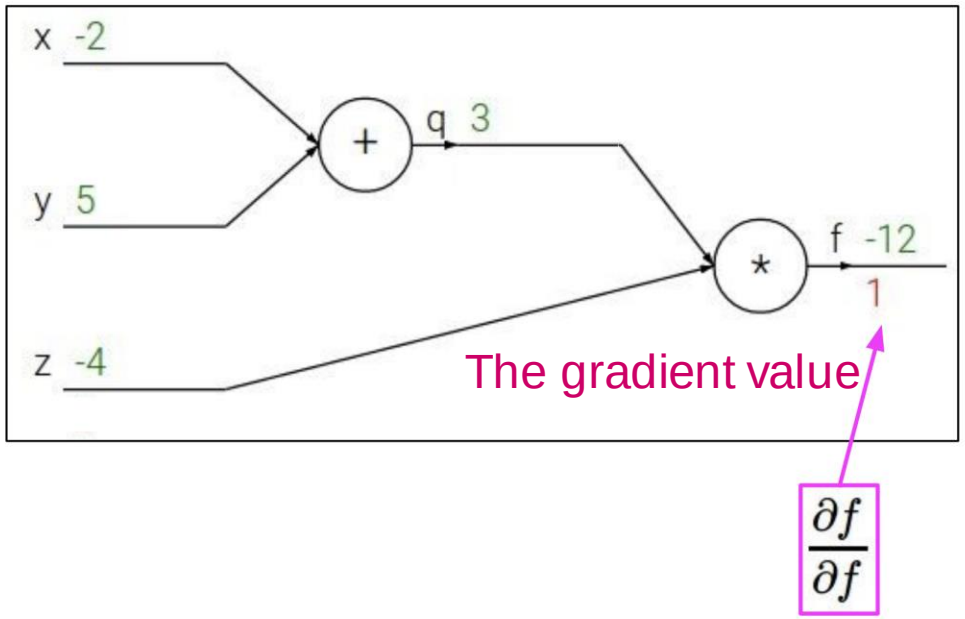
$$f(x, y, z) = (x + y)z$$

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



Let's start back prop!

Credit: Stanford CS231n

Back-propagation Example

Backpropagation: a simple example

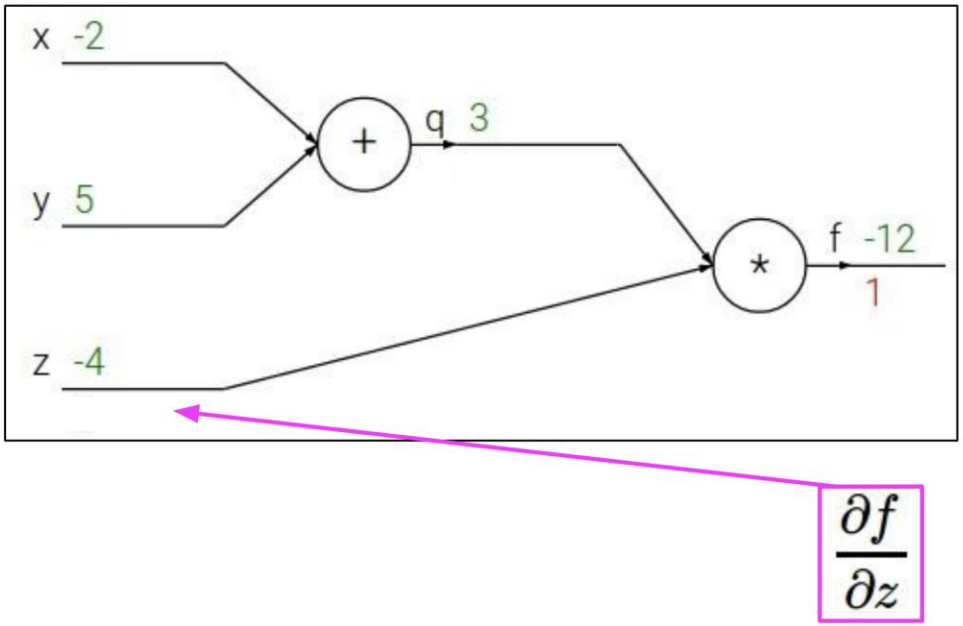
$$f(x, y, z) = (x + y)z$$

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



Back prop to z

Credit: Stanford CS231n

Back-propagation Example

Backpropagation: a simple example

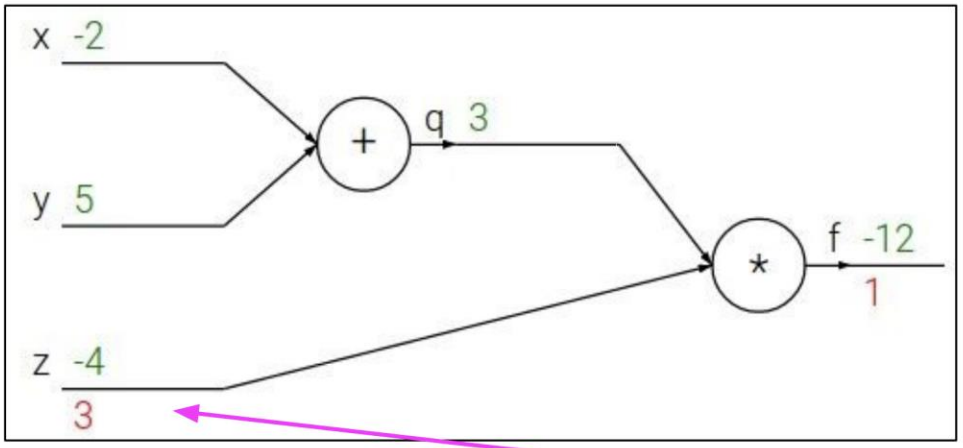
$$f(x, y, z) = (x + y)z$$

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

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

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

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



The gradient value

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

The gradient function that we need here

Back prop to z

Credit: Stanford CS231n

Back-propagation Example

Backpropagation: a simple example

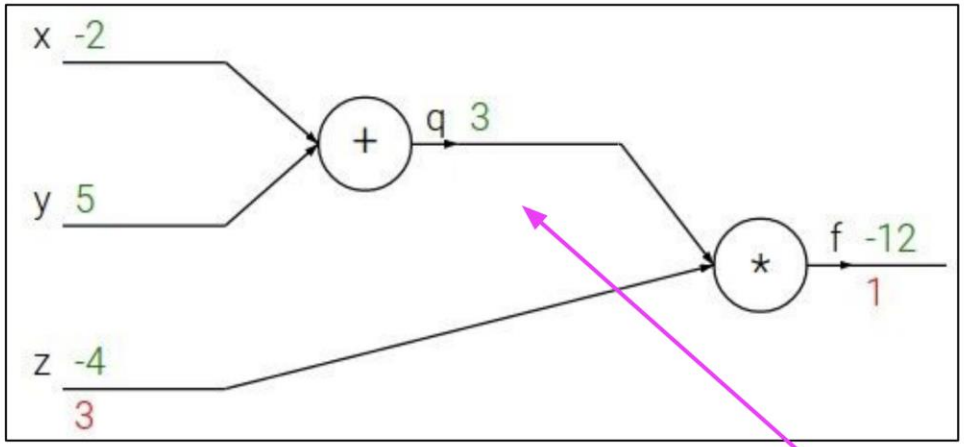
$$f(x, y, z) = (x + y)z$$

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



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

Back prop to q

Credit: Stanford CS231n

Back-propagation Example

Backpropagation: a simple example

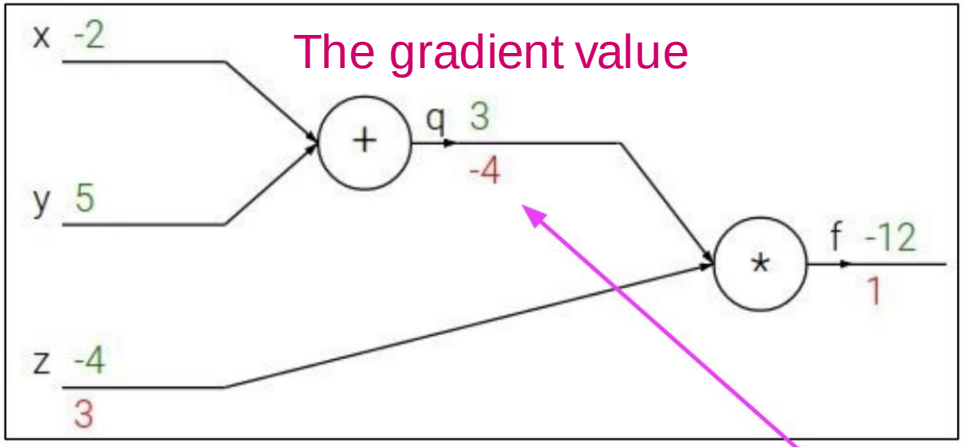
$$f(x, y, z) = (x + y)z$$

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



The gradient function that we need here

Credit: Stanford CS231n

Back-propagation Example

Backpropagation: a simple example

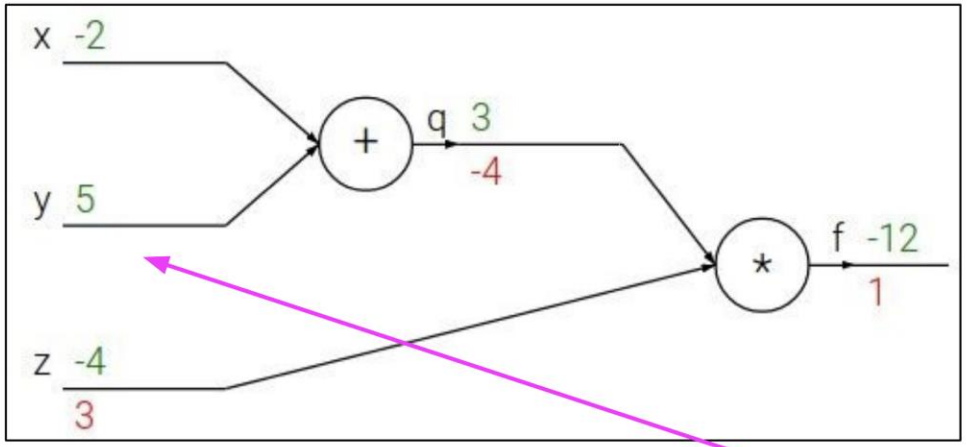
$$f(x, y, z) = (x + y)z$$

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



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

Chain rule:

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

Upstream gradient — This is -4

Local gradient — This is 1

Back prop to y

Credit: Stanford CS231n

Back-propagation Example

Backpropagation: a simple example

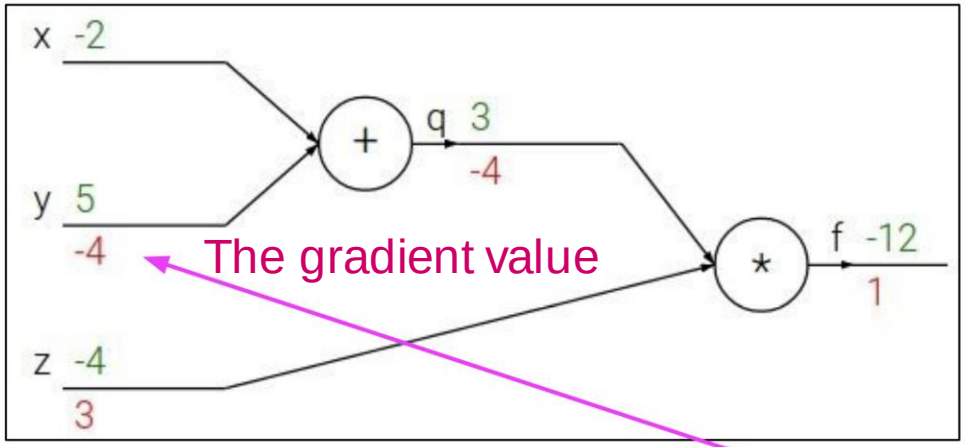
$$f(x, y, z) = (x + y)z$$

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



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

Chain rule:

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

Upstream gradient — This is -4

Local gradient — This is 1

Back prop to y

Credit: Stanford CS231n

Back-propagation Example

Backpropagation: a simple example

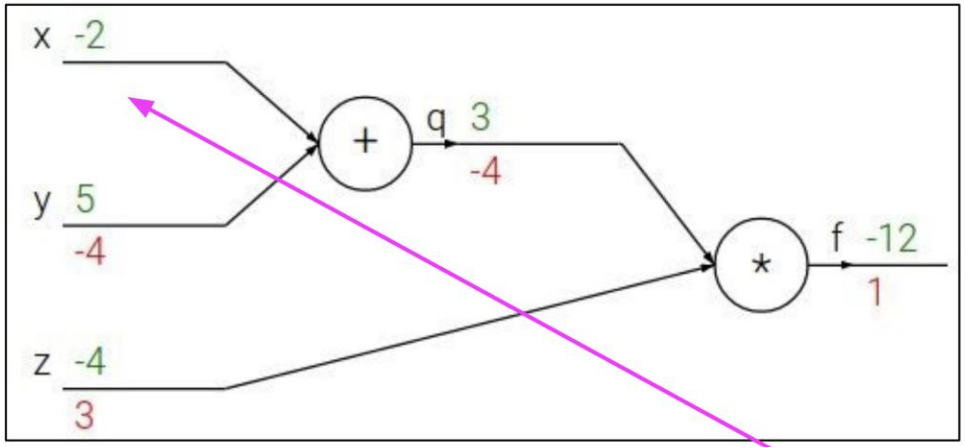
$$f(x, y, z) = (x + y)z$$

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 — This is -4

Local gradient — This is 1

Back prop to x

Credit: Stanford CS231n

Back-propagation Example

Backpropagation: a simple example

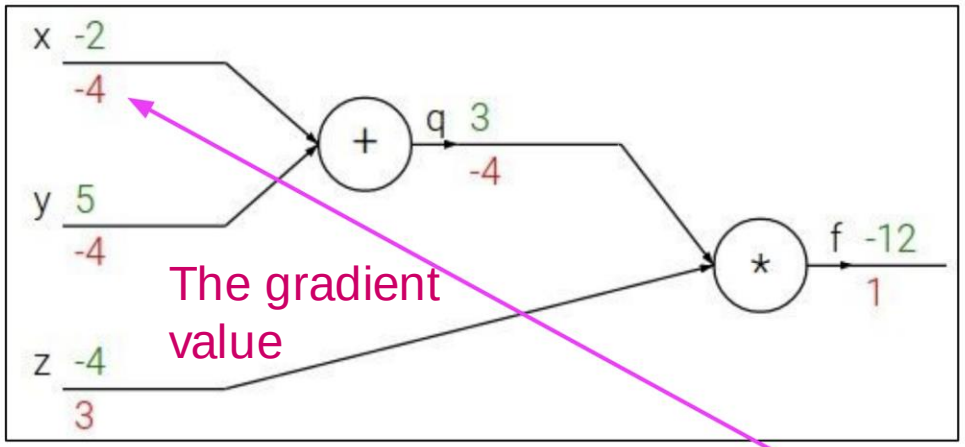
$$f(x, y, z) = (x + y)z$$

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 — This is -4

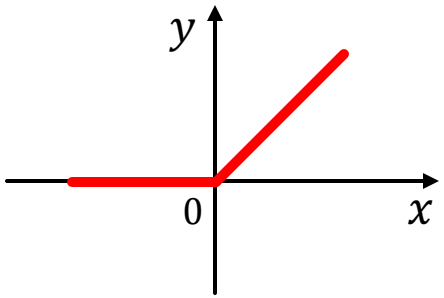
Local gradient — This is 1

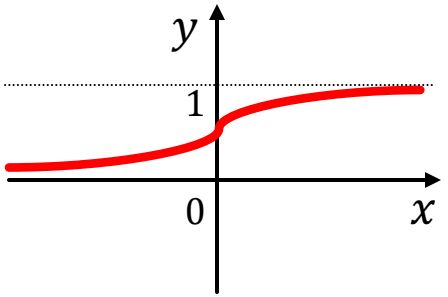
Back prop to x

Credit: Stanford CS231n

Non-linearity

- Note that in $f(\boldsymbol{x}) = W_2(W_1\boldsymbol{x})$, W_2W_1 is another matrix (vector, if we do binary classification)
- Hence $f(\boldsymbol{x})$ is still linear w.r.t. $\boldsymbol{x}$ no matter how many weight matrices we compose
- **Introduce non-linearity:**
  - **Rectified linear unit (ReLU)**
    $ReLU(x) = \max(x, 0)$
  - **Sigmoid**
    $$\sigma(x) = \frac{1}{1 + e^{-x}}$$



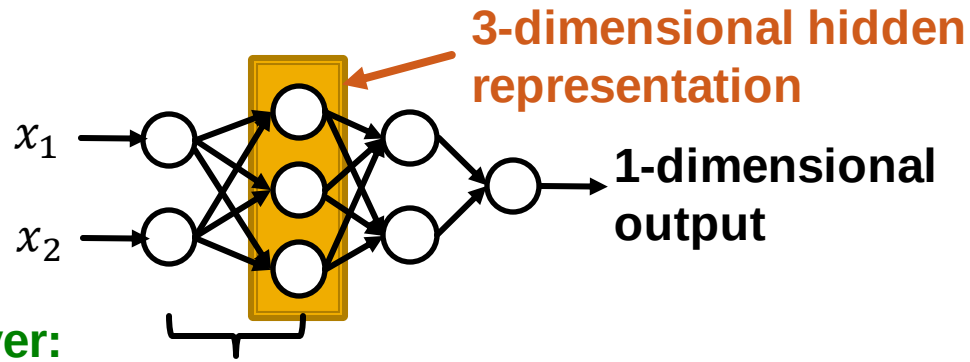


Multi-layer Perceptron (MLP)

- **Each layer of MLP combines linear transformation and non-linearity:**

$$\boldsymbol{x}^{(l+1)} = \sigma(W_l \boldsymbol{x}^{(l)} + b^l)$$

  - where W_l is weight matrix that transforms hidden representation at layer l to layer $l+1$
  - b^l is bias at layer l, and is added to the linear transformation of $\boldsymbol{x}$
  - σ is non-linearity function (e.g., sigmod)
- Suppose $\boldsymbol{x}$ is 2-dimensional, with entries x_1 and x_2



Summary

- **Objective function:**

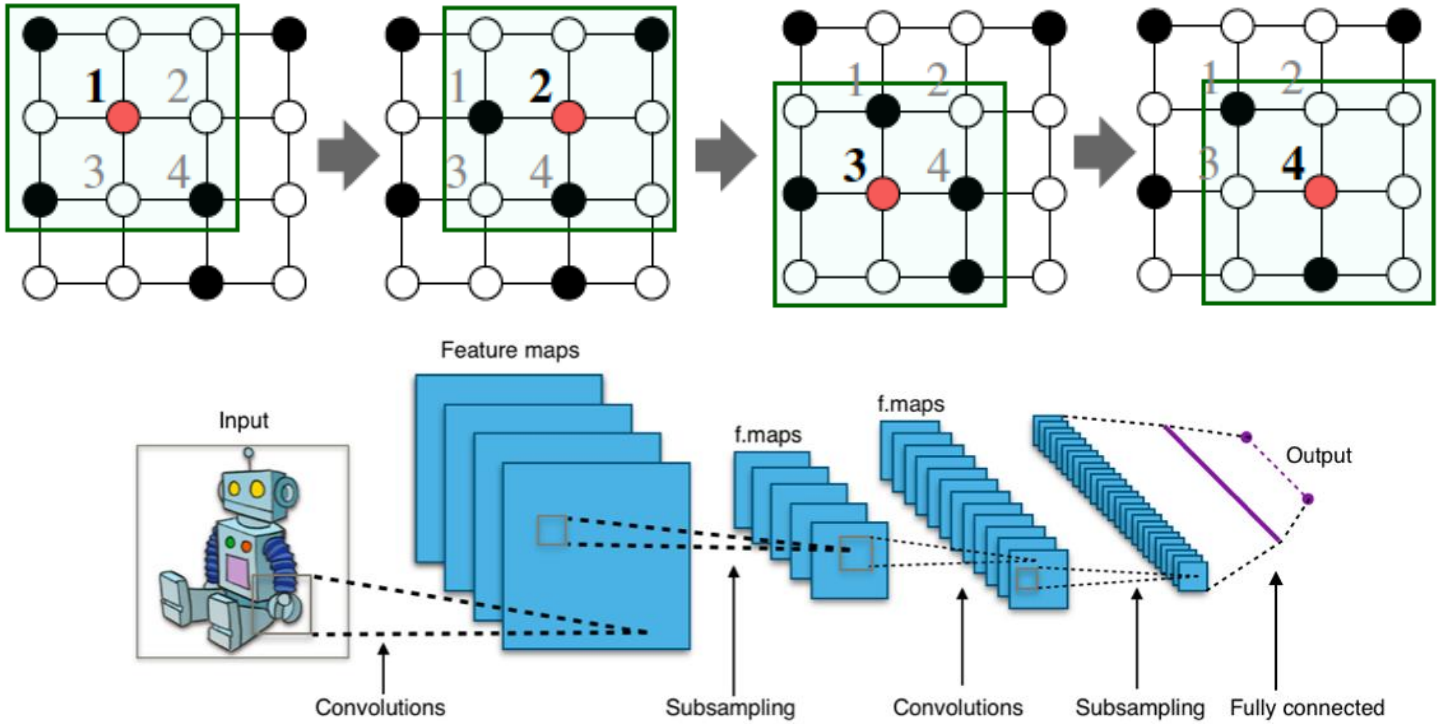
$$\min_{\Theta} \mathcal{L}(\boldsymbol{y}, f(\boldsymbol{x}))$$

- f can be a simple linear layer, an MLP, or other neural networks (e.g., a GNN later)
- Sample a minibatch of input $\boldsymbol{x}$
- **Forward propagation:** Compute $\mathcal{L}$ given $\boldsymbol{x}$
- **Back-propagation:** Obtain gradient $\nabla_{\Theta}\mathcal{L}$ using the chain rule
- Use **stochastic gradient descent (SGD)** to optimize for Θ over many iterations

Convolutional Neural Networks

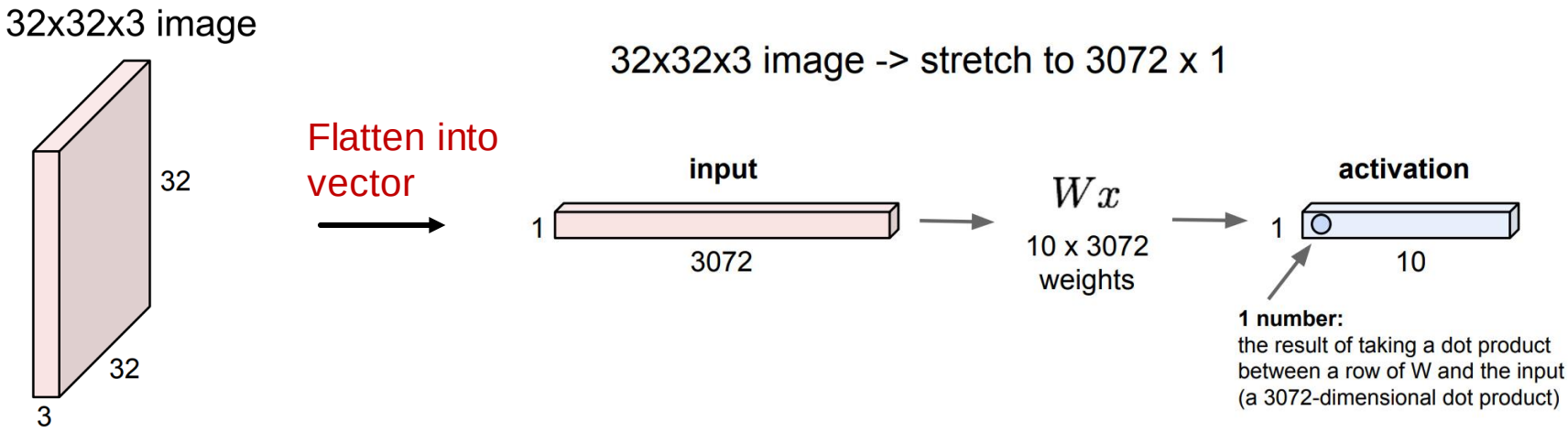
Convolutional Networks

CNN on an image:



A Naïve Approach

- Flatten an image into a vector
- Then apply a Linear layer $f(x) = Wx$



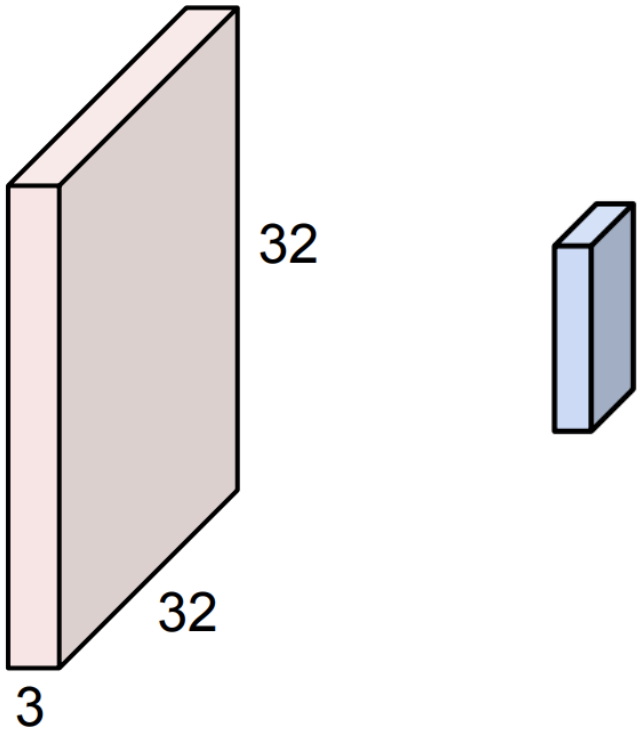
- **Issues**:
  - Cannot work with images with different sizes
  - Need $O(N^2D)$ parameters
    - N is the width/length of image, D is the number of channels

Credit: Stanford CS231n

Convolution Layer

Suppose we have RGB image (3 channels)

32*32*3 image 5*5*3 filter



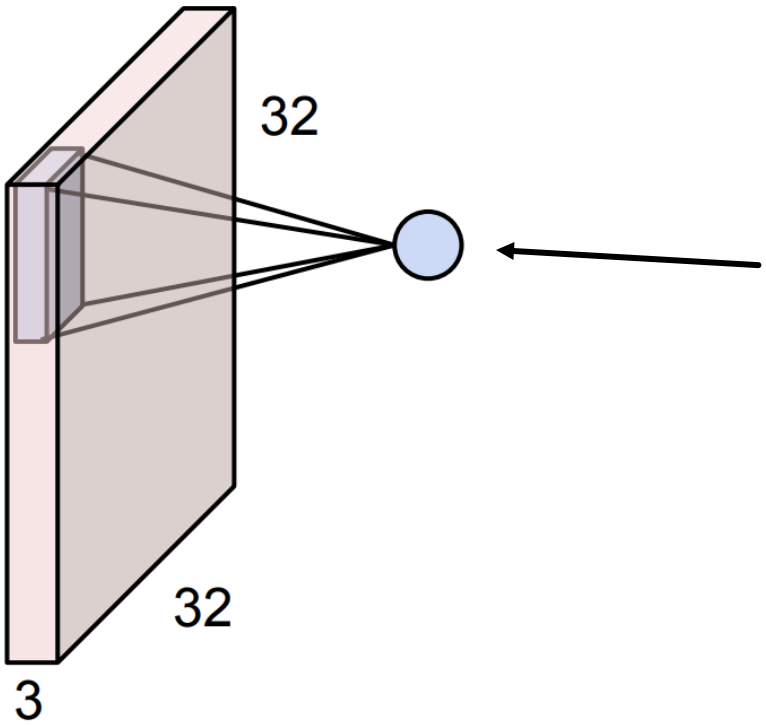
Convolve the filter with the image i.e. "**slide** over the image spatially, computing dot products"

Credit: Stanford CS231n

Convolution Layer

32*32*3 image

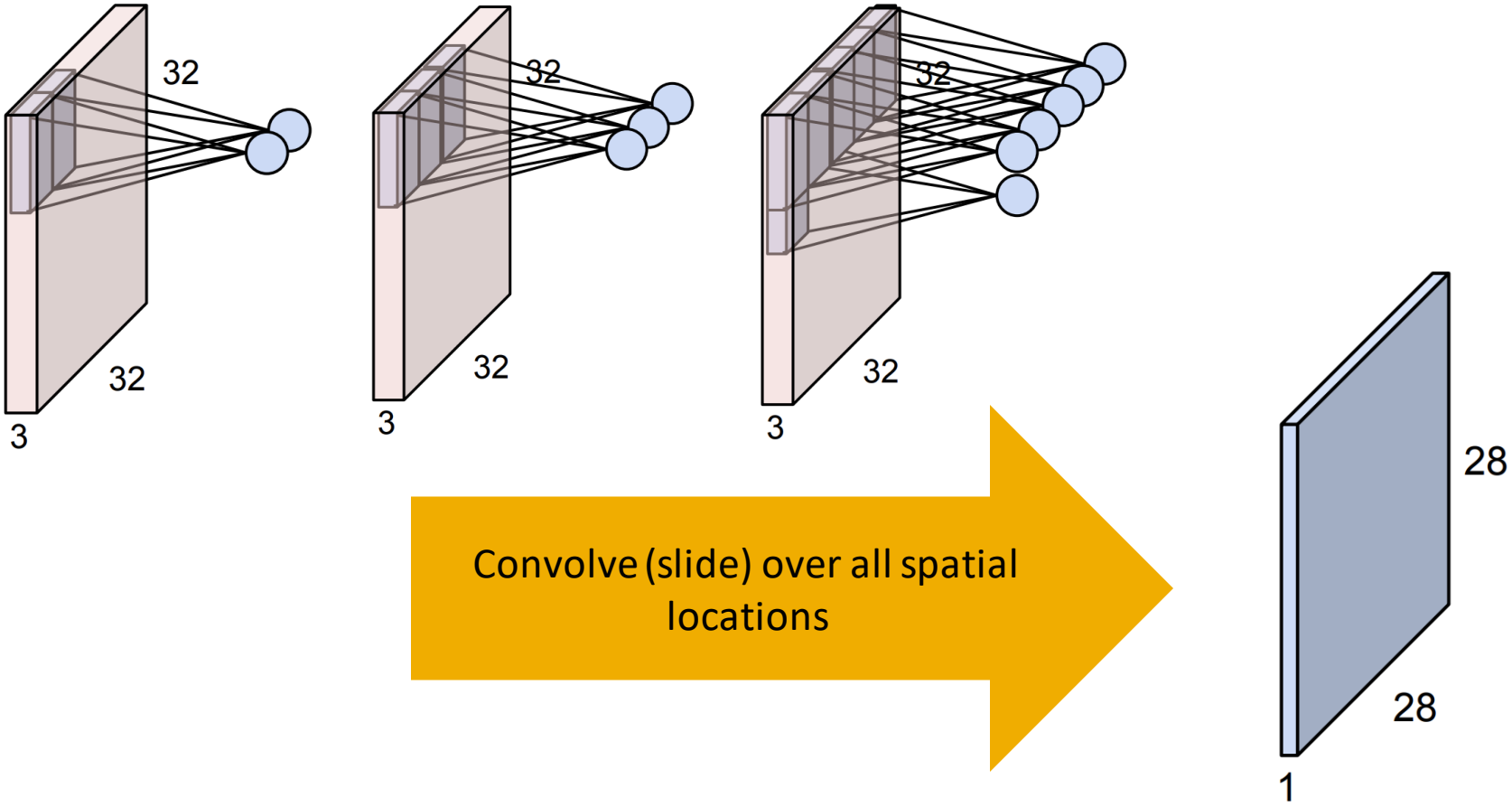
5*5*3 filter



The result of taking the dot product between the filter and a small 5*5*3 chunk of the image.
(i.e. 5*5*3 = 75-dimensional dot product + bias)

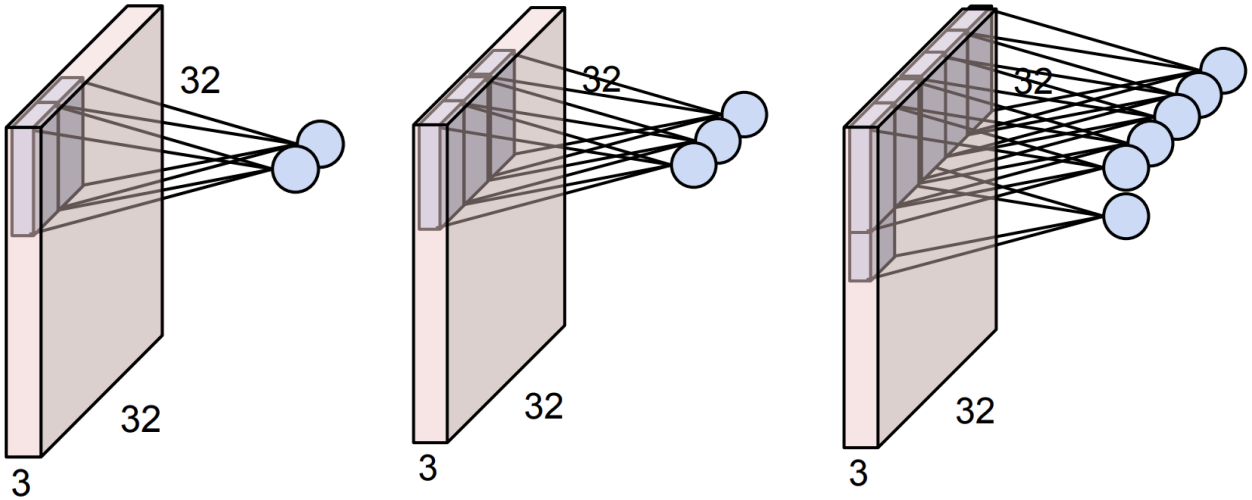
Credit: Stanford CS231n

Convolution Layer



28*28*1 output

Benefits of Convolution Layer



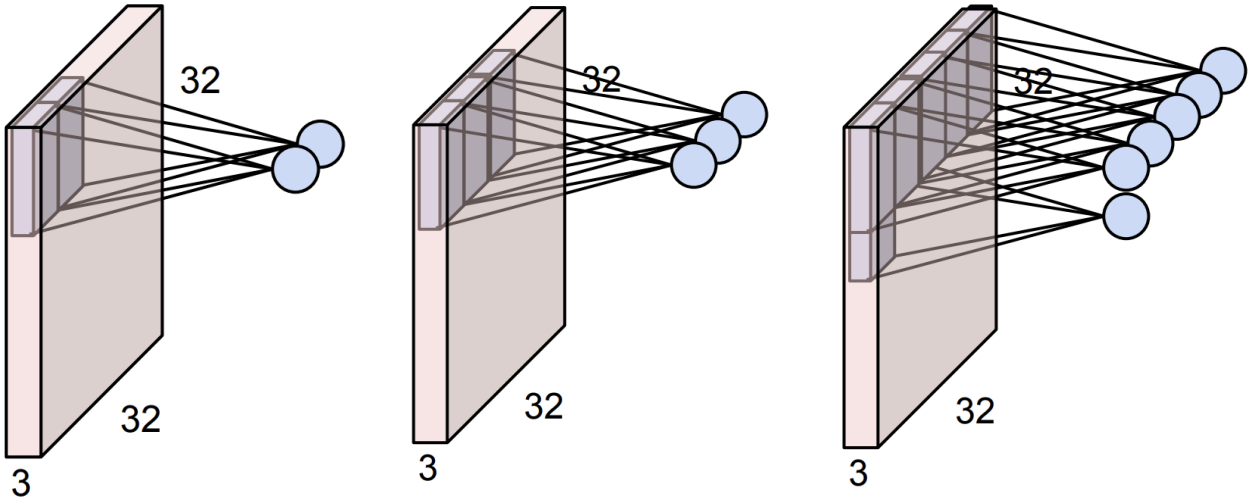
- **Solve the previous issues:**
  - The same Conv layer can work with images with different sizes
  - Only Need $O(K^2D)$ parameters rather than $O(N^2D)$

K: size of kernel

N: size of image

    - **5*5*3 parameters** vs **32*32*3 parameters**

Benefits of Convolution Layer



- **Solve the previous issues:**
  - The same Conv layer can work with images with different sizes
  - Only Need $O(K^2D)$ parameters rather than $O(N^2D)$
    - **5*5*3 parameters** vs **32*32*3 parameters**

K: size of kernel

N: size of image

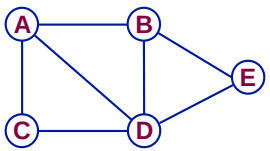
Graph Neural Networks

Setup: Learning from Graphs

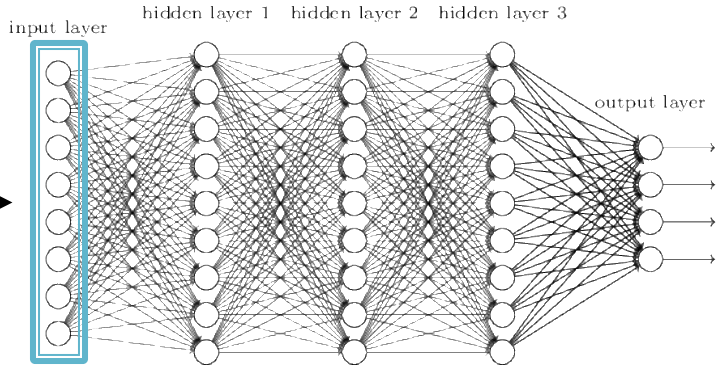
- **Assume we have a graph G:**
  - V is the **vertex set**
  - $\boldsymbol{A}$ is the **adjacency matrix** (assume binary)
  - $\boldsymbol{X} \in \mathbb{R}^{m \times |V|}$ is a matrix of **node features**
  - v: a node in V; $N(v)$: the set of neighbors of v.
  - **Node features:**
    - Social networks: User profile, User image
    - Biological networks: Gene expression profiles, gene functional information
    - When there is no node feature in the graph dataset:
      - Indicator vectors (one-hot encoding of a node)
      - Vector of constant 1: [1, 1, …, 1]

A Naïve Approach

- Join adjacency matrix and features
- Feed them into a deep neural net:



	A	B	C	D	E	Feat	
A	0	1	1	1	0	1	0
B	1	0	0	1	1	0	0
C	1	0	0	1	0	0	1
D	1	1	1	0	1	1	1
E	0	1	0	1	0	1	0

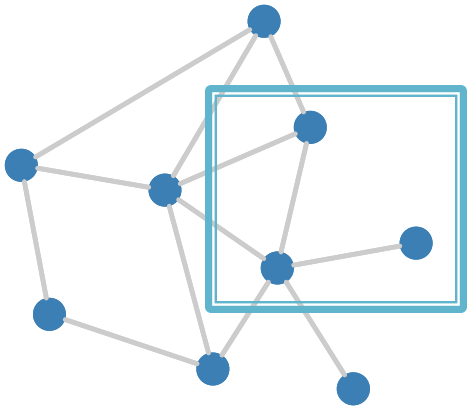


?

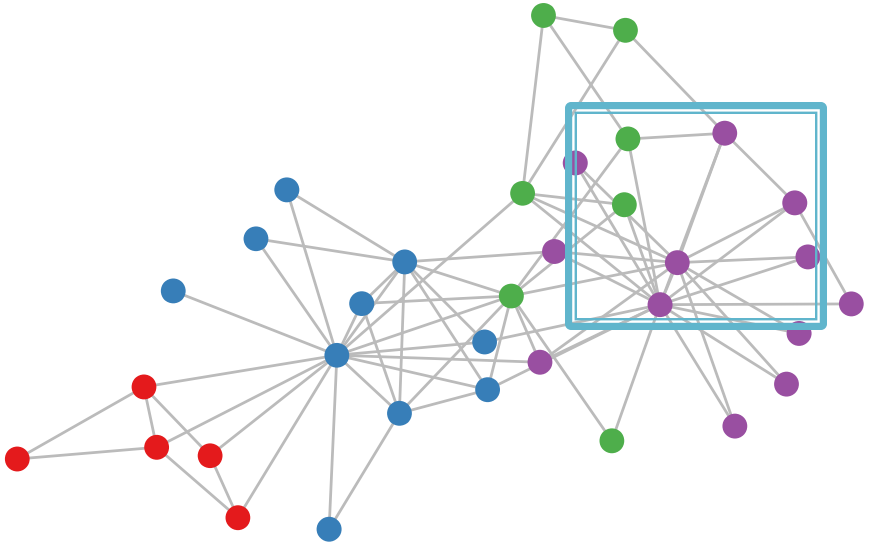
- Issues with this idea:
  - $O(|V|)$ parameters
  - Not applicable to graphs of different sizes
  - Sensitive to node ordering

Real-World Graphs

But our graphs look like this:



or this:

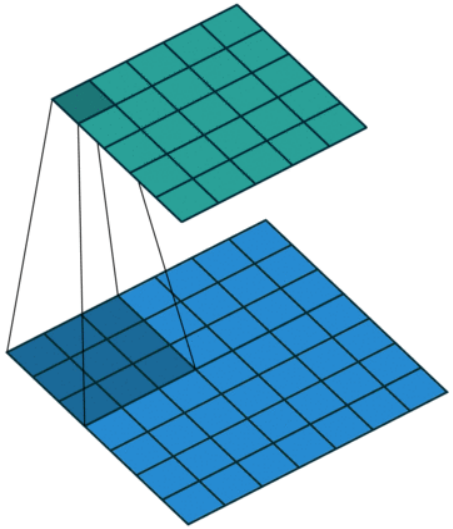


- There is no fixed notion of locality or sliding window on the graph
- Graph is permutation invariant

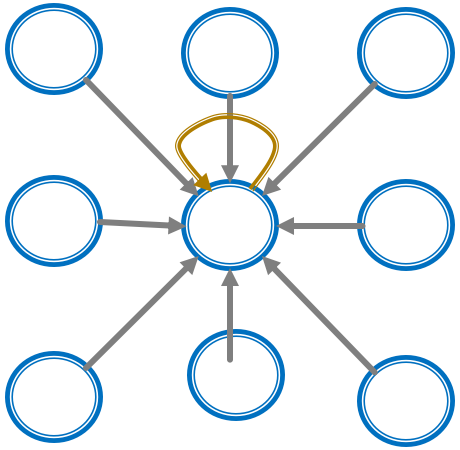
Credit: Stanford CS224W

From Images to Graphs

Single Convolutional neural network (CNN) layer with 3x3 filter:



Image



Graph

Idea: transform information at the neighbors and combine it:

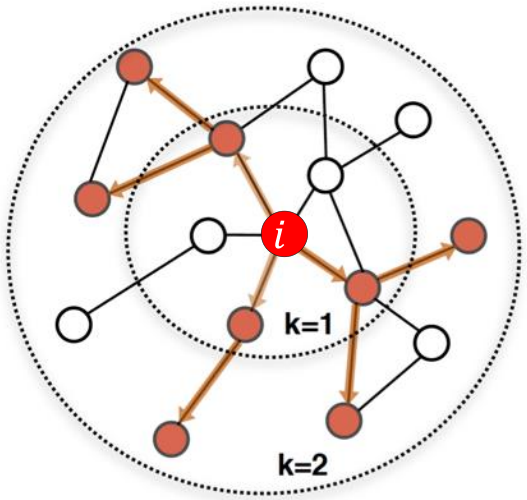
- Transform "messages" h_i from neighbors: $W_i\ h_i$
- Add them up: $\sum_i W_i\ h_i$

Credit: Stanford CS224W

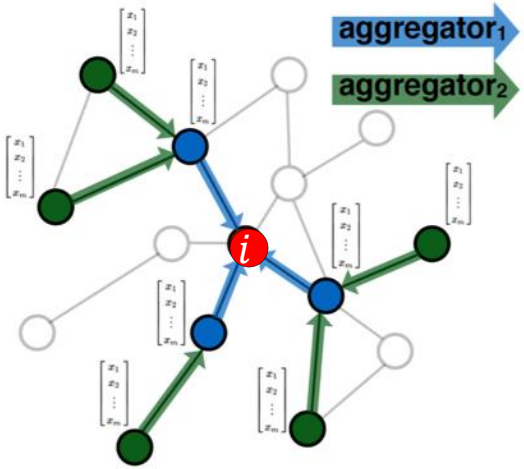
[Kipf and Welling, ICLR 2017]

Graph Convolutional Networks

Idea: Node's neighborhood defines a computation graph



Determine node computation graph



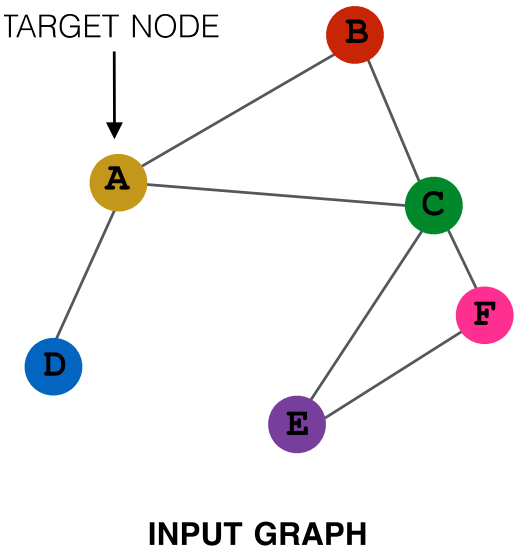
Propagate and transform information

Learn how to propagate information across the graph to compute node features

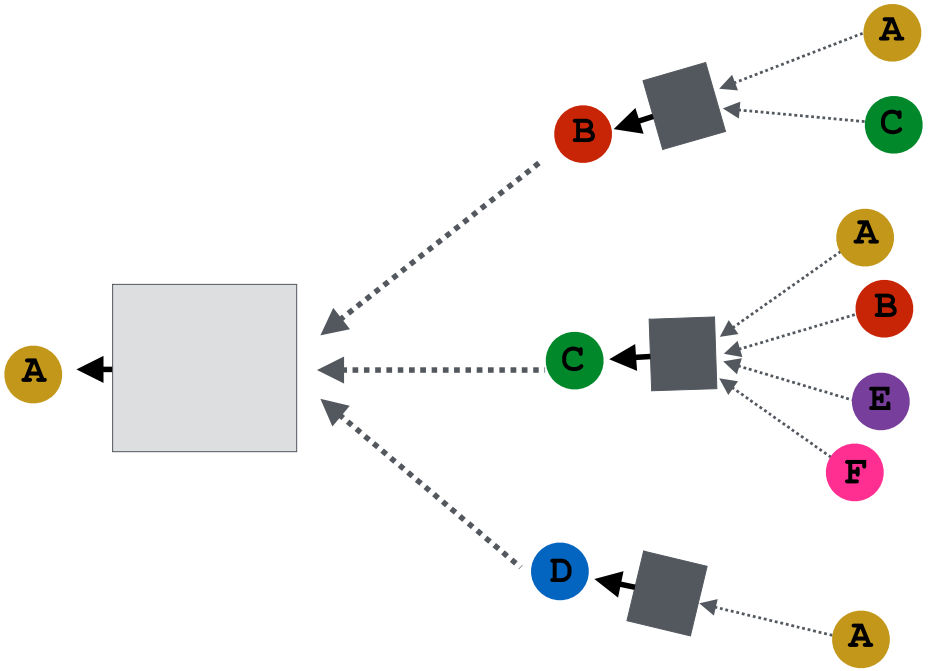
Credit: Stanford CS224W

Idea: Aggregate Neighbors

- **Key idea:** Generate node embeddings based on **local network neighborhoods**



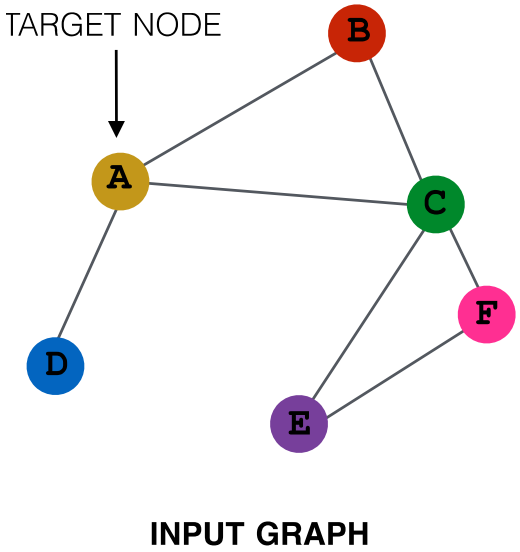
INPUT GRAPH



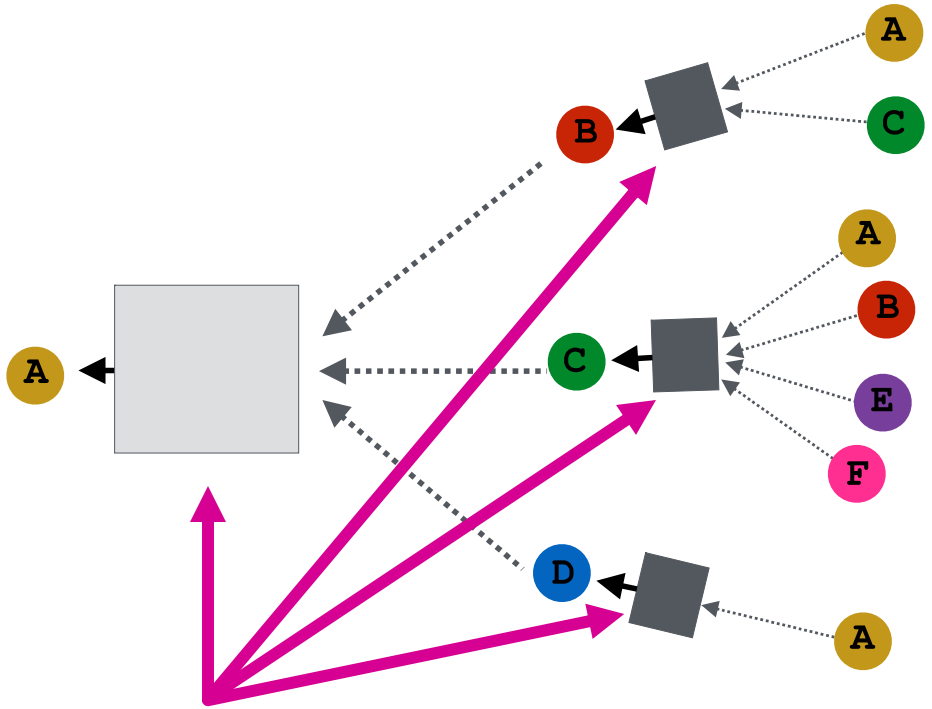
Credit: Stanford CS224W

Idea: Aggregate Neighbors

- **Intuition:** Nodes aggregate information from their neighbors using neural networks



INPUT GRAPH



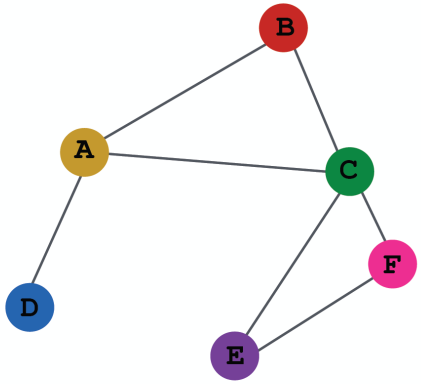
Neural networks

Credit: Stanford CS224W

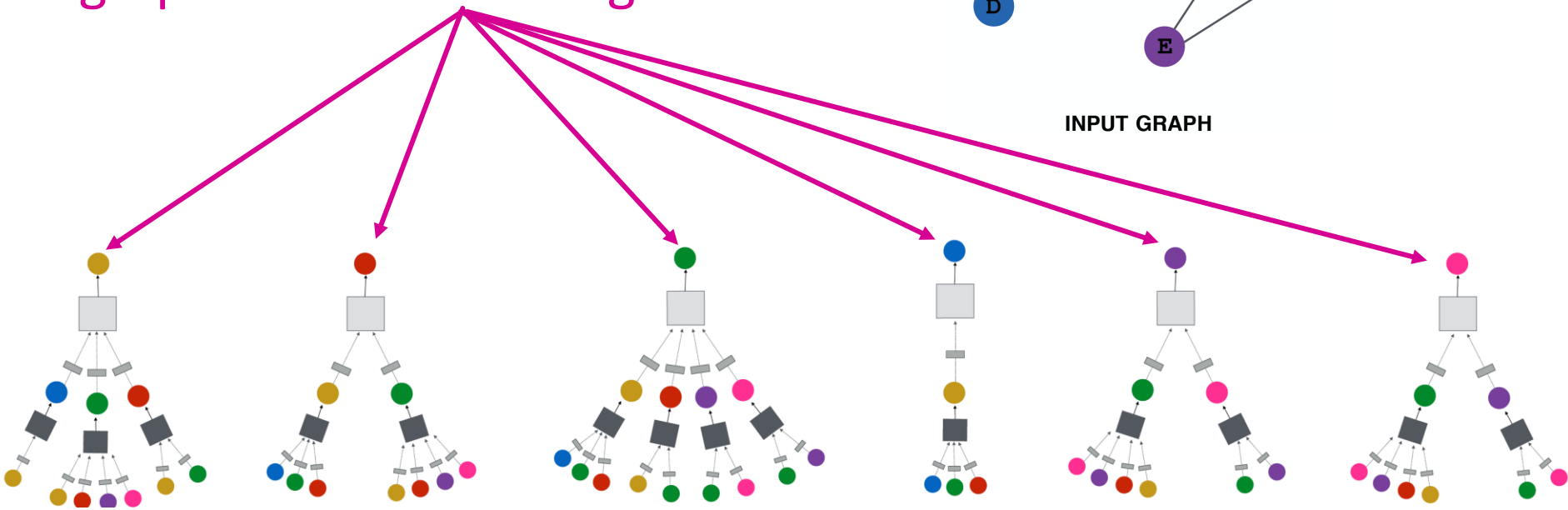
Idea: Aggregate Neighbors

- **Intuition:** Network neighborhood defines a computation graph

Every node defines a computation graph based on its neighborhood!



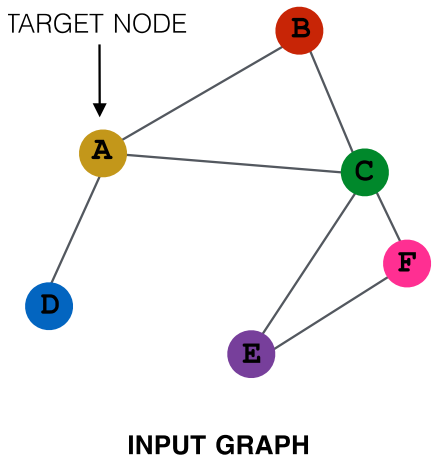
INPUT GRAPH

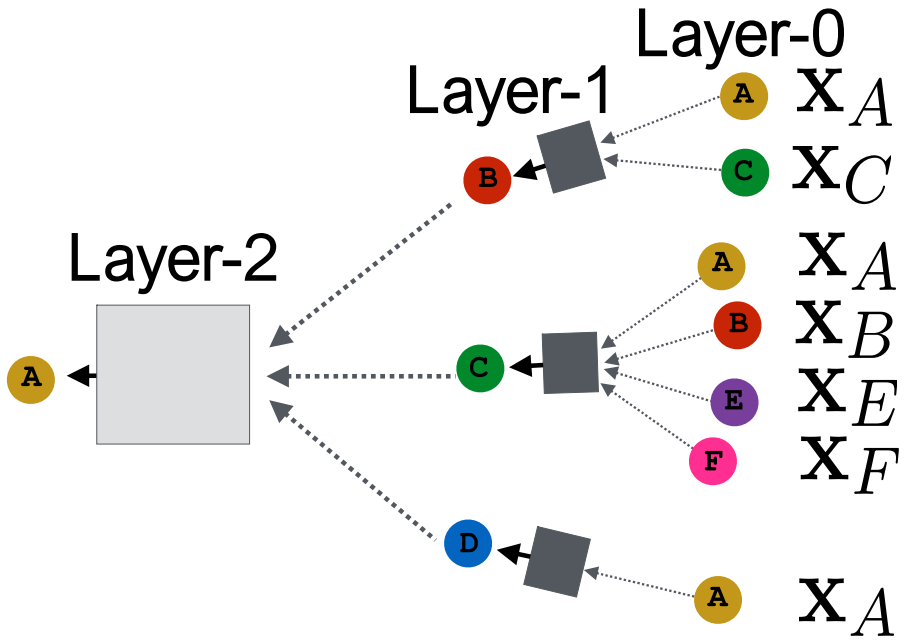


Credit: Stanford CS224W

Deep Model: Many Layers

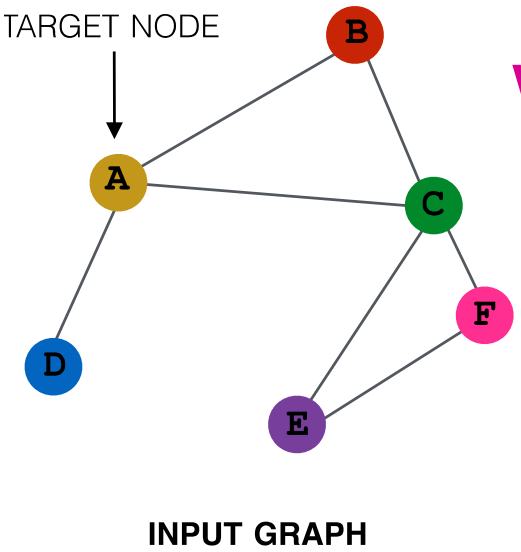
- Model can be of arbitrary depth:
  - Nodes have embeddings at each layer
  - Layer-0 embedding of node u is its input feature, x_u
  - Layer-k embedding gets information from nodes that are K hops away



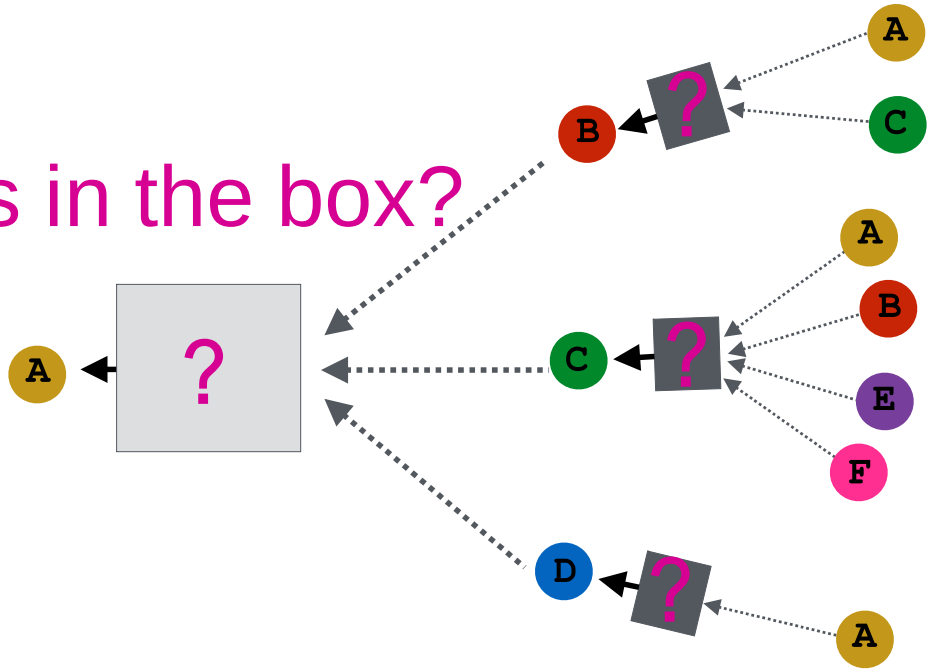


Neighborhood Aggregation

- **Neighborhood aggregation:** Key distinctions are in how different approaches aggregate information across the layers



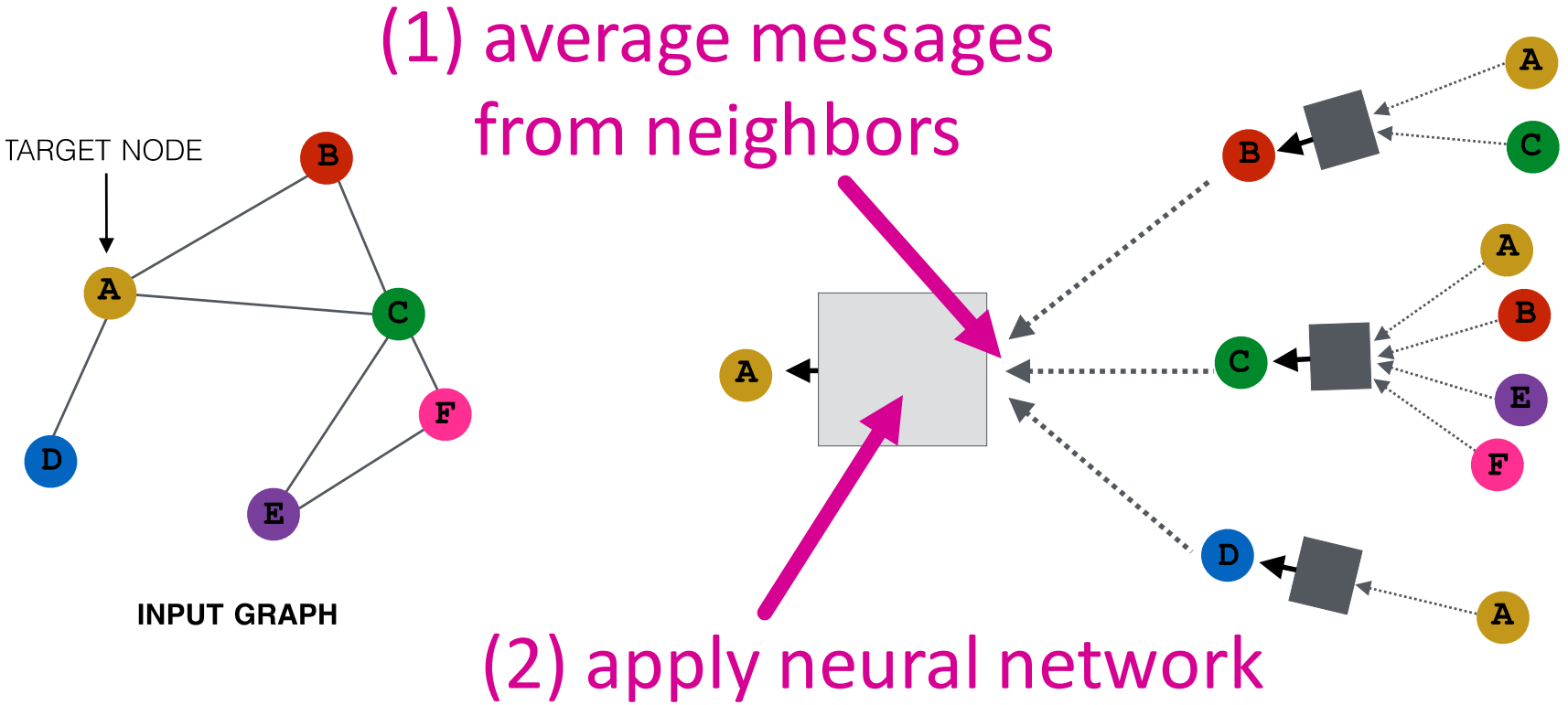
What is in the box?



Credit: Stanford CS224W

Neighborhood Aggregation

- **Basic approach:** Average information from neighbors and apply a neural network



Credit: Stanford CS224W

The Math: Deep Encoder

- **Basic approach:** Average neighbor messages and apply a neural network

Initial 0-th layer embeddings are equal to node features

$$\mathrm{h}_v^0 = \mathrm{x}_v$$

embedding of v at layer l

$$\mathrm{h}_v^{(l+1)} = \sigma\left(\mathrm{W}_l \sum_{u \in \mathrm{N}(v)} \frac{\mathrm{h}_u^{(l)}}{|\mathrm{N}(v)|} + \mathrm{B}_l \mathrm{h}_v^{(l)}\right), \forall l \in \{0, \ldots, L-1\}$$

$$\mathrm{z}_v = \mathrm{h}_v^{(L)}$$

Embedding after L layers of neighborhood aggregation

Non-linearity (e.g., ReLU)

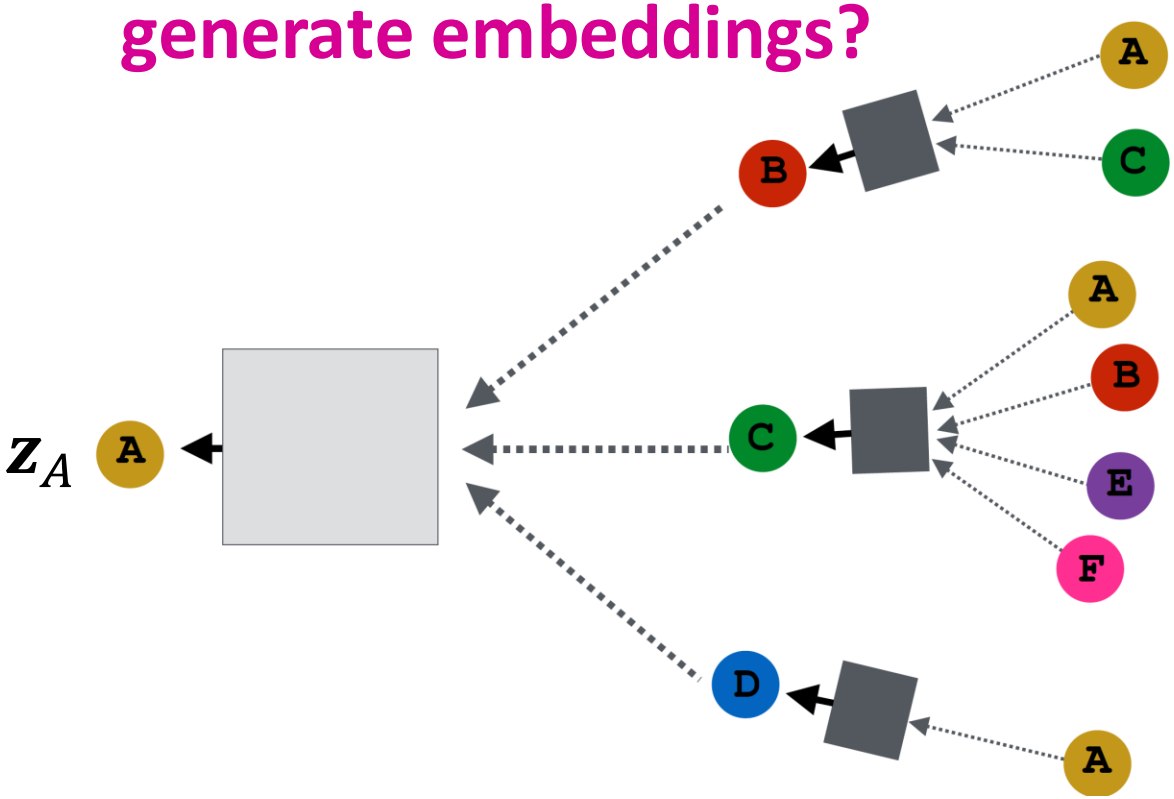
Average of neighbor's previous layer embeddings

Total number of layers

Credit: Stanford CS224W

Training the Model

How do we train the model to generate embeddings?



Need to define a loss function on the embeddings

Credit: Stanford CS224W

Model Parameters

Trainable weight matrices
(i.e., what we learn)

$$\mathrm{h}_v^{(0)} = \mathrm{x}_v$$

$$\mathrm{h}_v^{(l+1)} = \sigma\left(\mathrm{W}_l \sum_{u \in \mathrm{N}(v)} \frac{\mathrm{h}_u^{(l)}}{|\mathrm{N}(v)|} + \mathrm{B}_l \mathrm{h}_v^{(l)}\right), \forall l \in \{0, \ldots, L-1\}$$

$$\mathrm{z}_v = \mathrm{h}_v^{(L)}$$

Final node embedding

We can feed these **embeddings into any loss function** and run SGD to **train the weight parameters**

h_v^l: the hidden representation of node v at layer l

- W_k: weight matrix for neighborhood aggregation
- B_k: weight matrix for transforming hidden vector of self

Credit: Stanford CS224W

How to train a GNN

- Node embedding $\boldsymbol{z}_v$ is a function of input graph
- **Supervised setting**: we want to minimize the loss $\mathcal{L}$ (see also slide 15):

$$\min_{\Theta} \mathcal{L}(\boldsymbol{y}, f(\boldsymbol{z}_v))$$

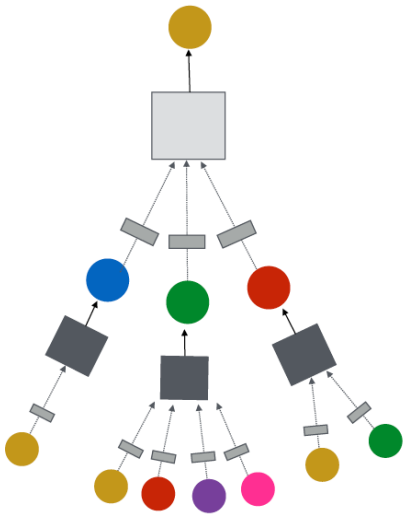
  - $\boldsymbol{y}$: node label
  - $\mathcal{L}$ could be L2 if $\boldsymbol{y}$ is real number, or cross entropy if $\boldsymbol{y}$ is categorical

Credit: Stanford CS224W

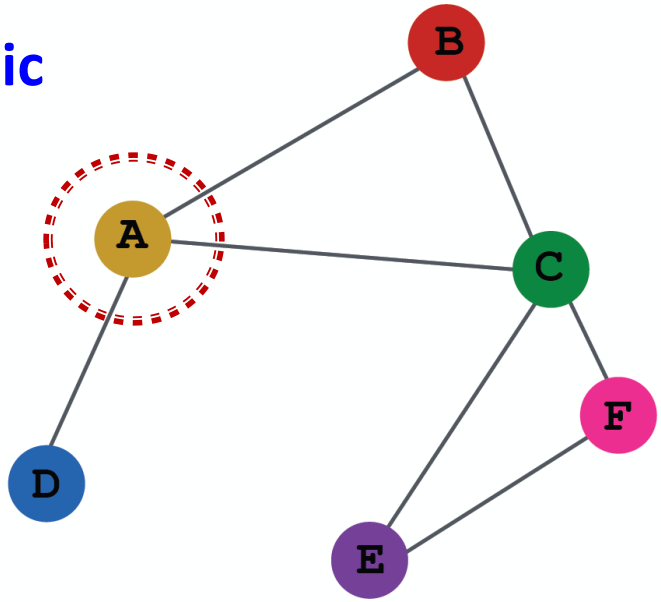
Supervised Training

Directly train the model for a supervised task (e.g., node classification)

Safe or toxic drug?



Safe or toxic drug?



E.g., a drug-drug interaction network

Supervised Training

Directly train the model for a supervised task (e.g., **node classification**)

- Use cross entropy loss (slide 16)

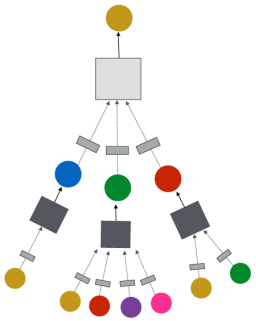
$$\mathcal{L} = \sum_{v \in V} y_v \log(\sigma(z_v^T \theta)) + (1 - y_v) \log(1 - \sigma(z_v^T \theta))$$

Encoder output: node embedding

Classification weights

Node class label

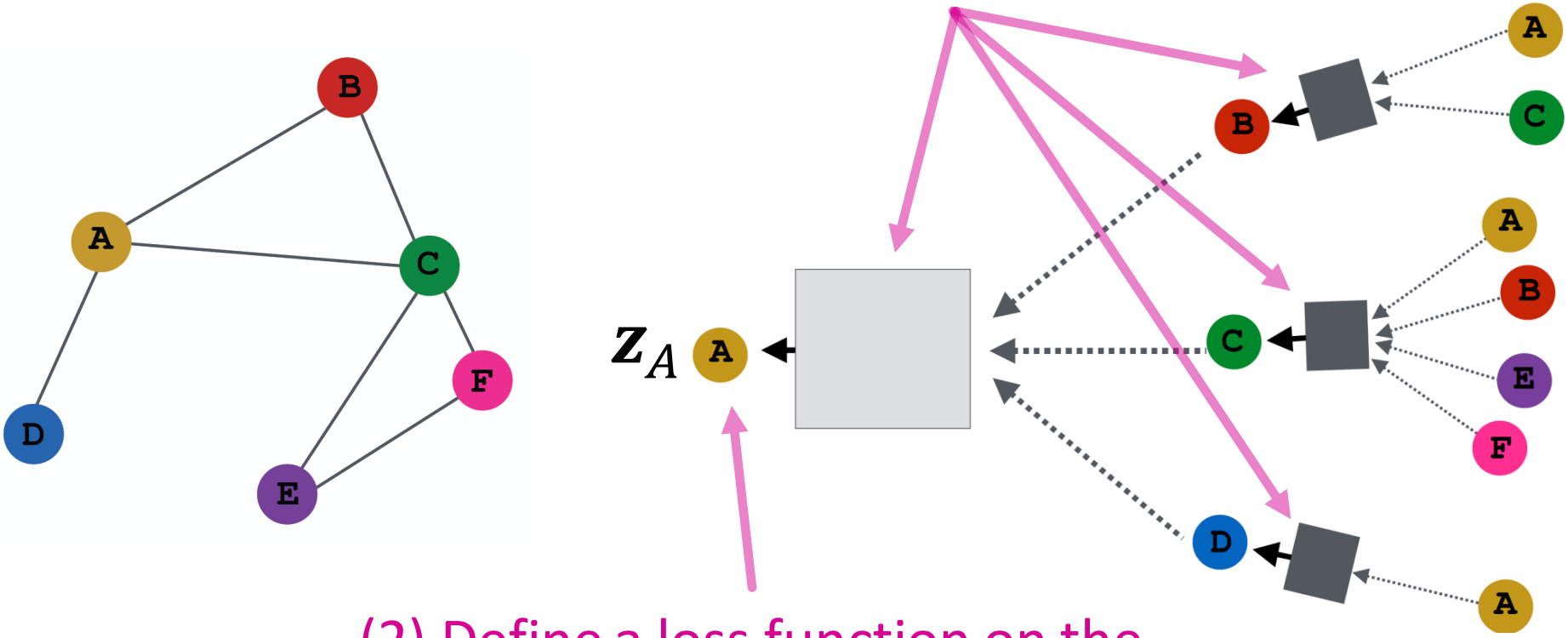
Safe or toxic drug?



Credit: Stanford CS224W

Model Design: Overview

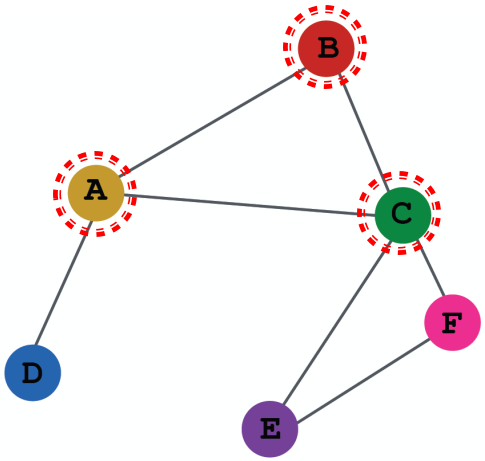
(1) Define a neighborhood aggregation function



(2) Define a loss function on the embeddings

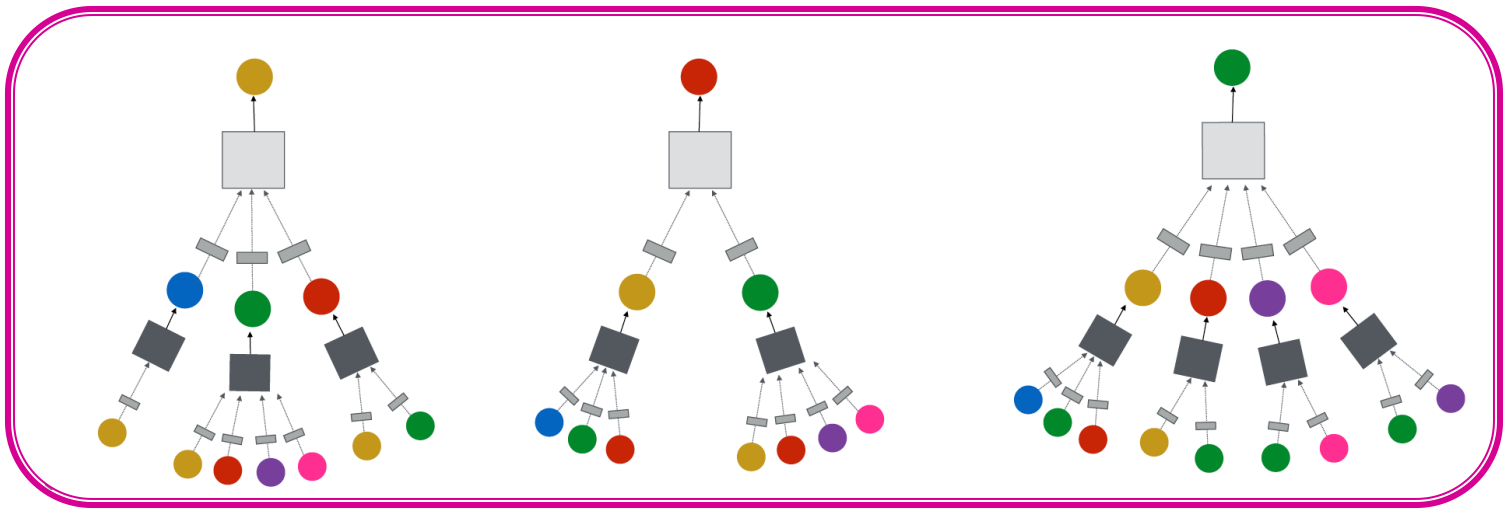
Credit: Stanford CS224W

Model Design: Overview

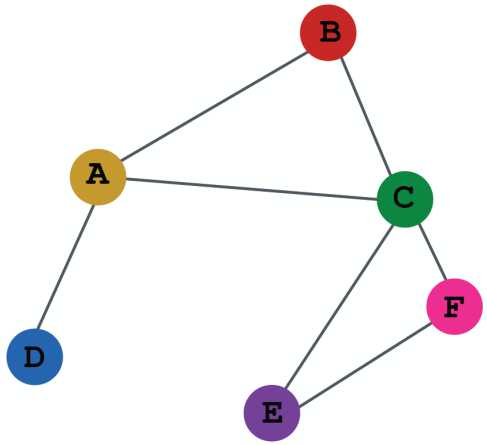


INPUT GRAPH

(3) Train on a set of nodes, i.e., a batch of compute graphs



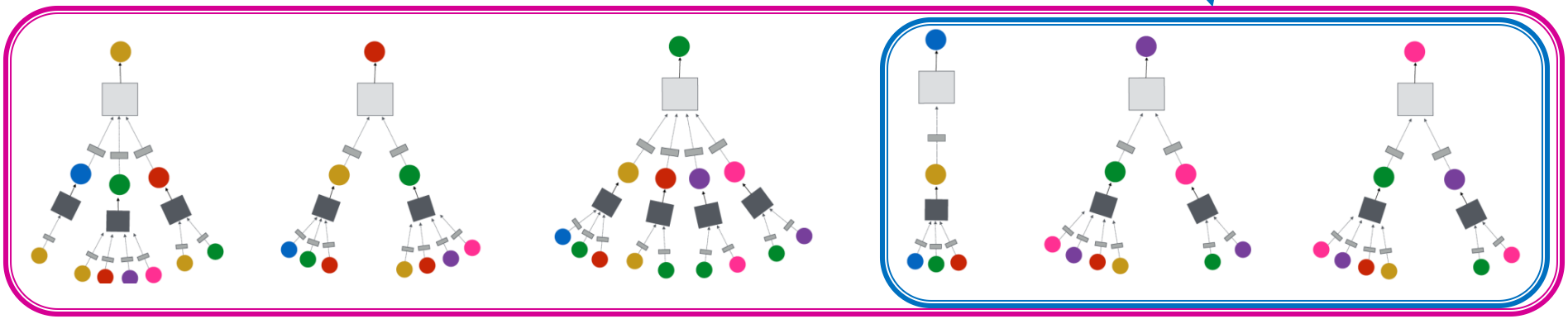
Model Design: Overview



INPUT GRAPH

(4) Generate embeddings for nodes as needed

Even for nodes we never trained on!



Summary of This Lecture

- **The key considerations for ML:**
  - The model $f_\theta(x)$: Linear model, neural networks, …
  - The loss $\mathcal{L}$: Classification & regression losses
  - Optimize the loss: Backpropagation
- **Two widely used Deep Learning models**
  - Convolutional Neural Networks
  - Graph Neural Networks