DSC250: Advanced Data Mining

Graph Neural Networks

Zhiting Hu

Lecture 13, November 9, 2023

UC San Diego
HALICIOĞLU DATA SCIENCE INSTITUTE

Outline

- Graph Neural Networks (GNNs)

- 4 paper presentations
  - Swetha Mamidipoodi, Tanvi Joshi
  - Giorgia Nicolaou, Lia Ran
  - Yuheng Zha, Yuwei Zhang
  - Pengcen Jiang, Zhongyang Zhang

Graph Neural Networks (GNNs)

Slides adapted from:

- Jure Leskovec, Stanford CS224W: Machine Learning with Graphs

Deep Graph Encoders

- Encoding based on graph neural networks

$$\text{ENC}(v) =$$ **multiple layers of non-linear transformations based on graph structure**

v.s. Shallow Encoder:

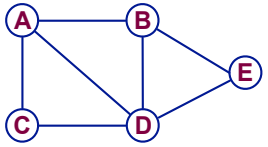
$$\text{ENC}(v) = \mathbf{z}_v = \mathbf{Z} \cdot v$$

Graph Neural Networks: Setup

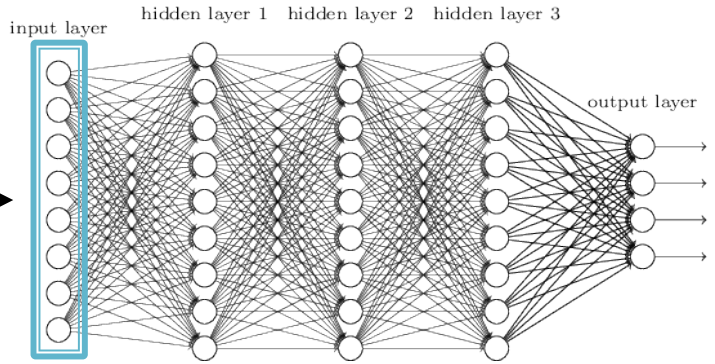
- **Assume we have a graph G:**
  - V is the **vertex set**
  - $\boldsymbol{A}$ is the **adjacency matrix** (assume binary)
  - $\boldsymbol{X} \in \mathbb{R}^{|V| \times d}$ is a matrix of **node features**
  - v: a node in V; $N(v)$: the set of neighbors of v.
  - **Node features:**
    - Social networks: 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



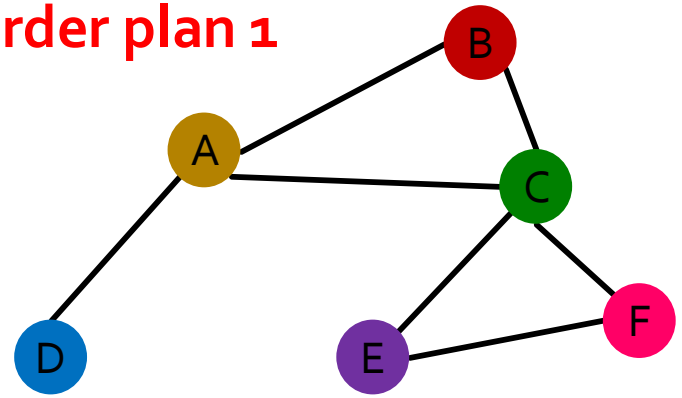
?

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

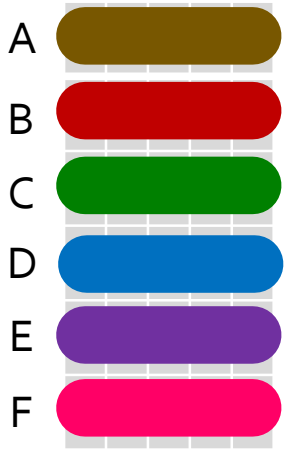
Permutation Invariance

- **Graph does not have a canonical order of the nodes!**

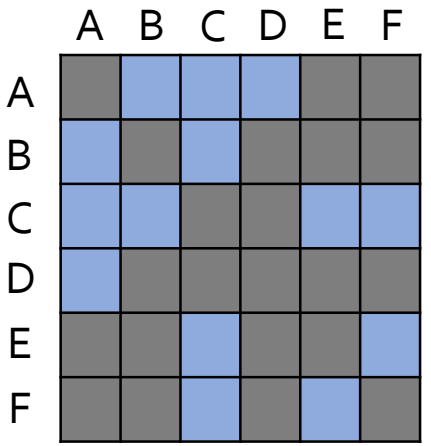
Order plan 1



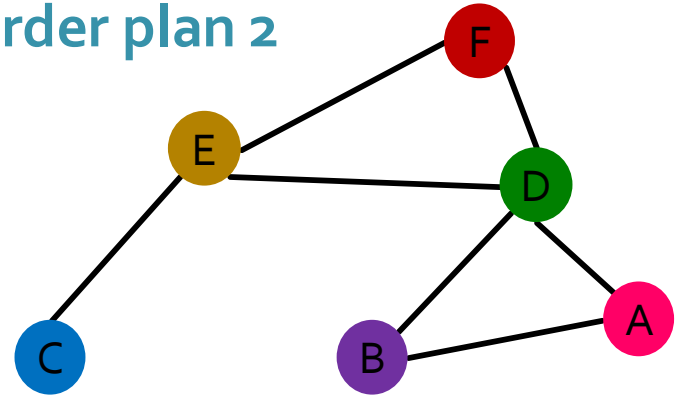
Node features X_1



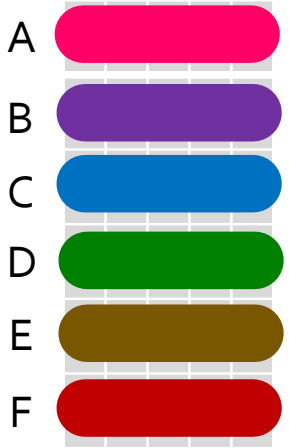
Adjacency matrix A_1



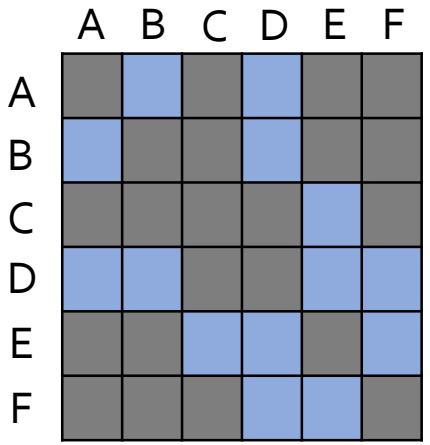
Order plan 2



Node features X_2

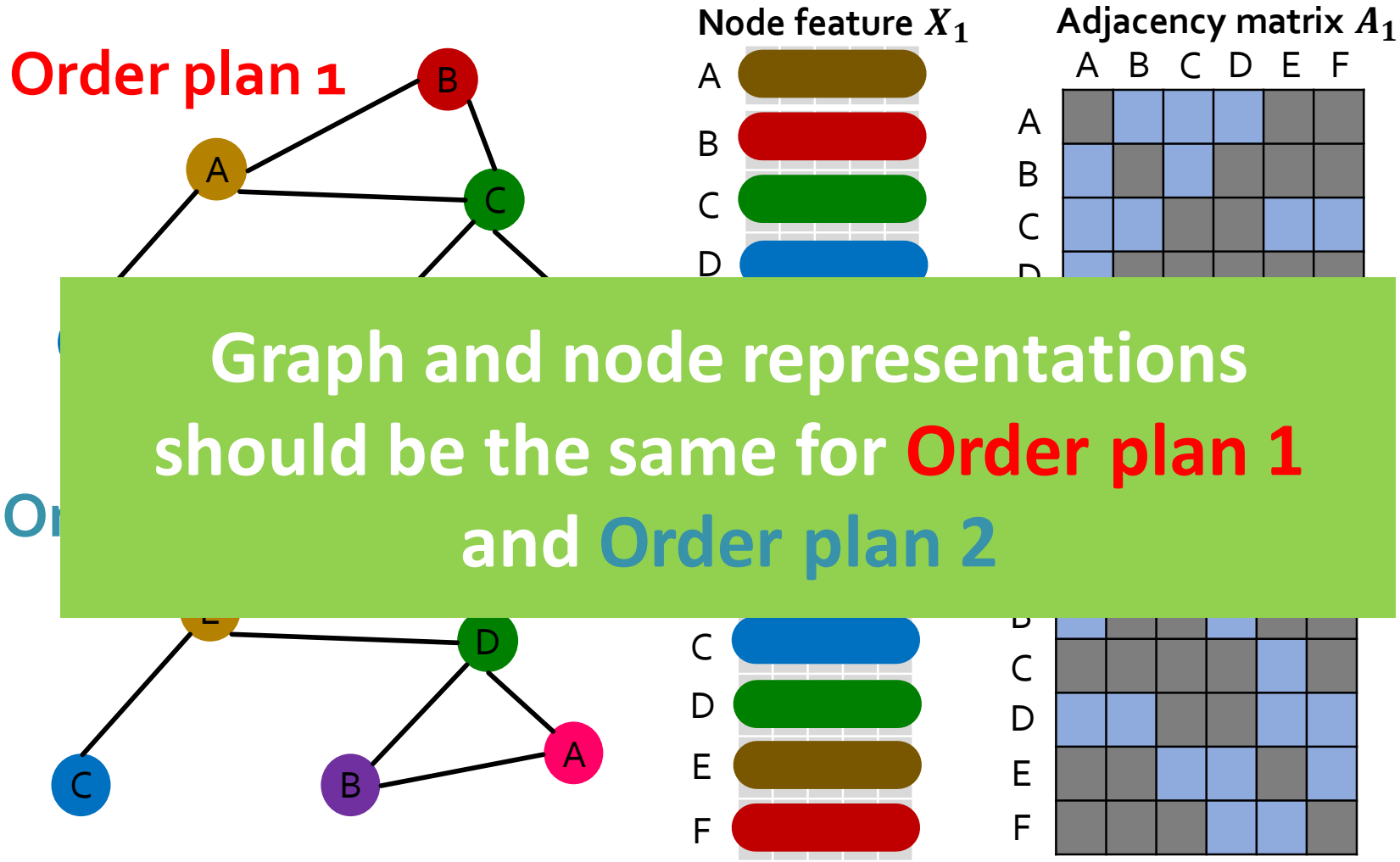


Adjacency matrix A_2



Permutation Invariance

- **Graph does not have a canonical order of the nodes!**



Permutation Invariance

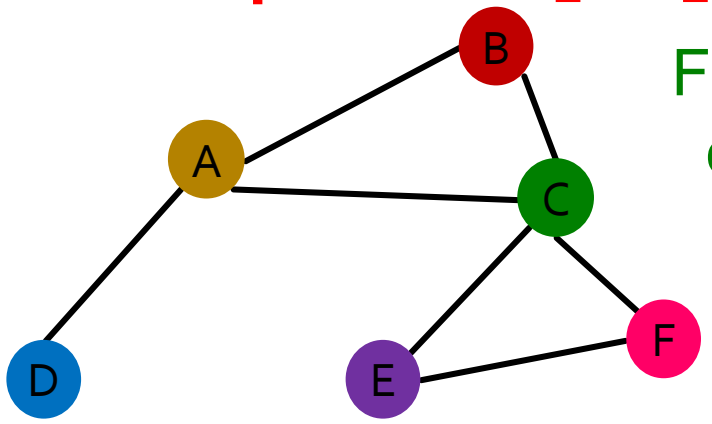
What does it mean by "graph representation is same for two order plans"?

- Consider we learn a function f that maps a graph $G = (\boldsymbol{A}, \boldsymbol{X})$ to a vector $\mathbb{R}^d$ then

$$f(\boldsymbol{A}_1, \boldsymbol{X}_1) = f(\boldsymbol{A}_2, \boldsymbol{X}_2)$$

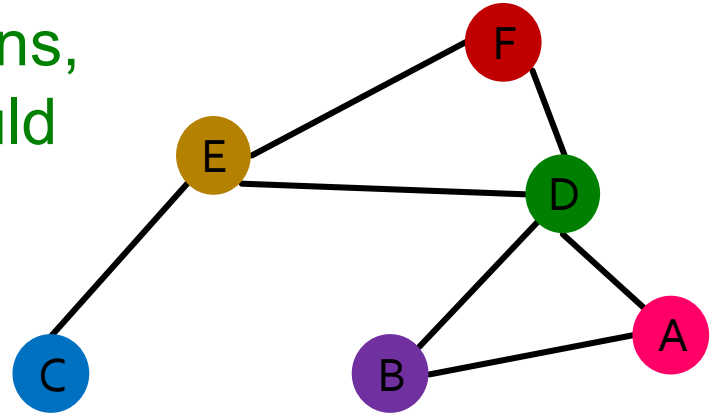
$\boldsymbol{A}$ is the adjacency matrix
$\boldsymbol{X}$ is the node feature matrix

Order plan 1: $\boldsymbol{A}_1, \boldsymbol{X}_1$



For two order plans, output of f should be the same!

Order plan 2: $\boldsymbol{A}_2, \boldsymbol{X}_2$



Permutation Invariance

What does it mean by "graph representation is same for two order plans"?

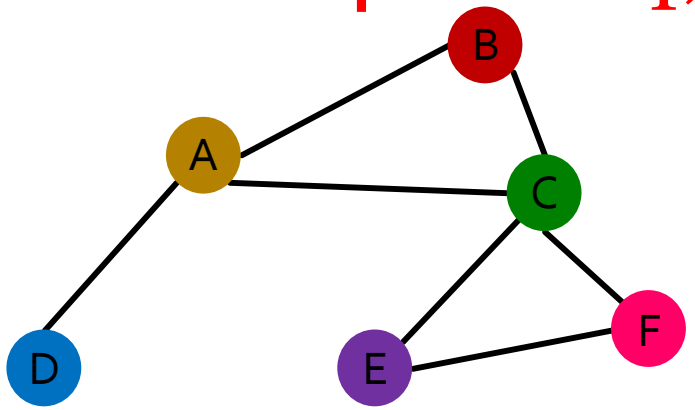
- Consider we learn a function f that maps a graph $G = (\boldsymbol{A}, \boldsymbol{X})$ to a vector $\mathbb{R}^d$.
  $\boldsymbol{A}$ is the adjacency matrix
  $\boldsymbol{X}$ is the node feature matrix
- Then, if $f(\boldsymbol{A}_i, \boldsymbol{X}_i) = f(\boldsymbol{A}_j, \boldsymbol{X}_j)$ for any order plan i and j, we formally say f is a **permutation invariant function**.
  For a graph with $|V|$ nodes, there are $|V|!$ different order plans.
- **Definition:** For any graph function f: $\mathbb{R}^{|V|\times m} \times \mathbb{R}^{|V|\times|V|} \to \mathbb{R}^d$, f is **permutation-invariant** if $f(A, X) = f(PAP^T, PX)$ for any permutation P.

Permutation P: a shuffle of the node order
Example: (A,B,C)->(B,C,A)

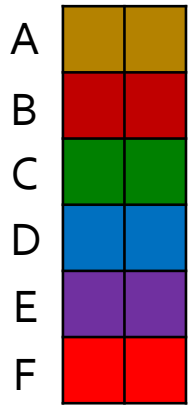
Permutation Equivariance

For node representation: We learn a function f that maps nodes of G to a matrix $\mathbb{R}^{m \times d}$.

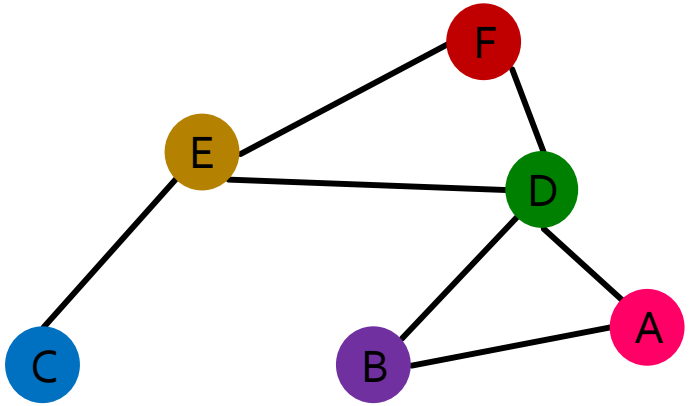
Order plan 1: A_1, X_1



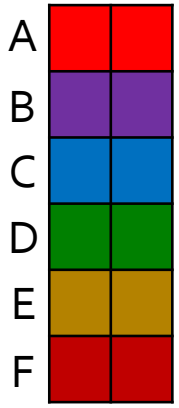
$$f(\boldsymbol{A}_1, \boldsymbol{X}_1) =$$



Order plan 2: A_2, X_2



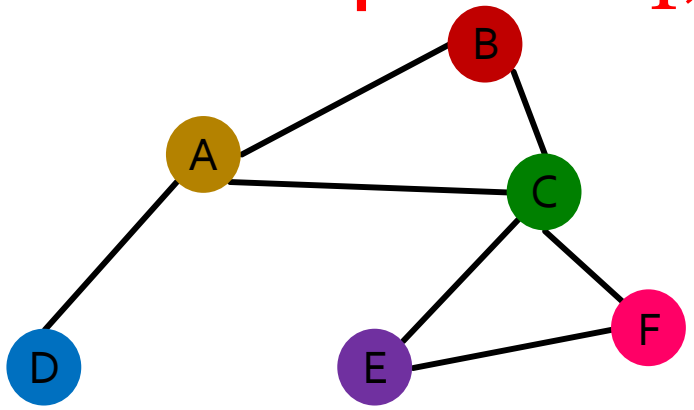
$$f(\boldsymbol{A}_2, \boldsymbol{X}_2) =$$



Permutation Equivariance

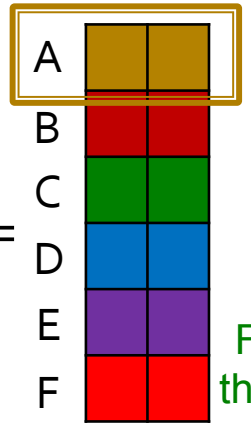
For node representation: We learn a function f that maps nodes of G to a matrix $\mathbb{R}^{m \times d}$.

Order plan 1: A_1, X_1

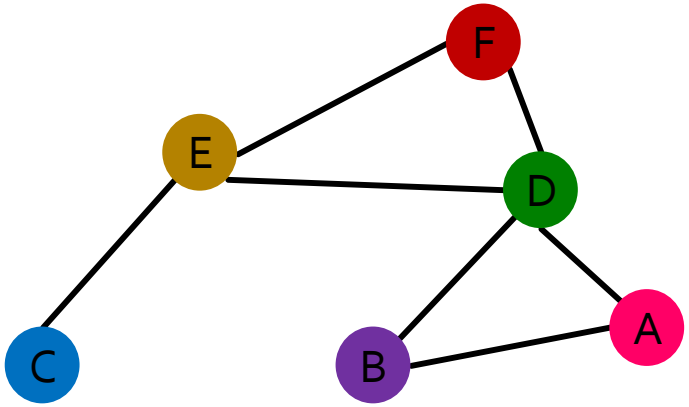


Representation vector of the brown node A

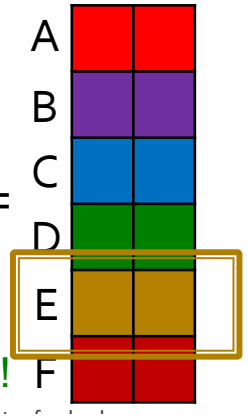
$$f(A_1, X_1) =$$



Order plan 2: A_2, X_2



$$f(A_2, X_2) =$$



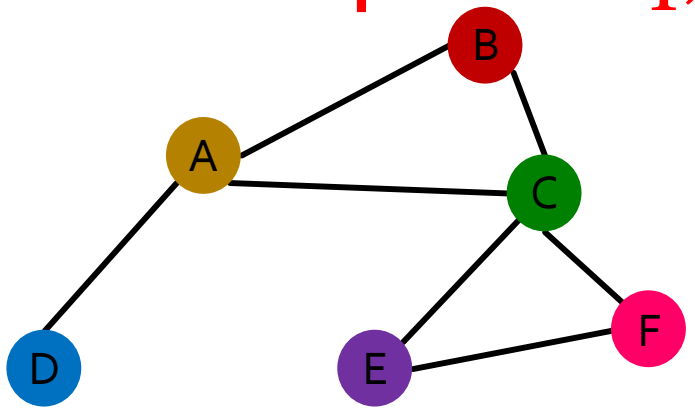
Representation vector of the brown node E

For two order plans, the vector of node at the same position in the graph is the same!

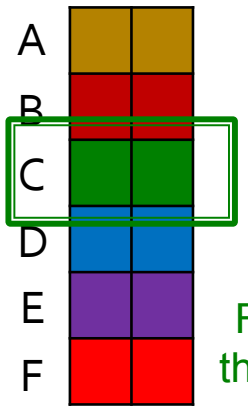
Permutation Equivariance

For node representation: We learn a function f that maps nodes of G to a matrix $\mathbb{R}^{m \times d}$.

Order plan 1: A_1, X_1

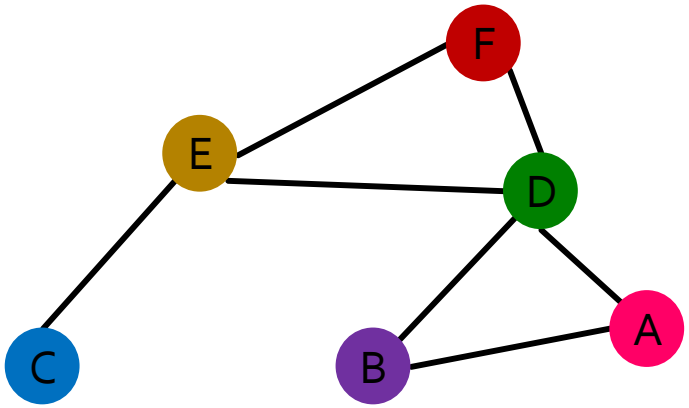


$f(A_1, X_1) =$

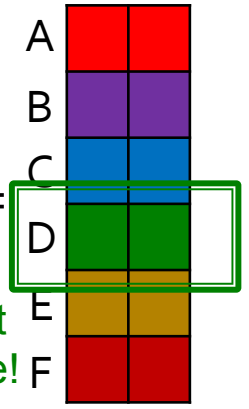


Representation vector of the green node C

Order plan 2: A_2, X_2



$f(A_2, X_2) =$



Representation vector of the green node D

For two order plans, the vector of node at the same position in the graph is the same!

Permutation Equivariance

For node representation

- Consider we learn a function f that maps a graph $G = (\boldsymbol{A}, \boldsymbol{X})$ to a matrix $\mathbb{R}^{m \times d}$
- If the output vector of a node at the same position in the graph remains unchanged for any order plan, we say f is **permutation equivariant**.
- **Definition:** For any node function f: $\mathbb{R}^{|V| \times m} \times \mathbb{R}^{|V| \times |V|} \rightarrow \mathbb{R}^{|V| \times m}$, f is **permutation-equivariant** if $Pf(A, X) = f(PAP^T, PX)$ for any permutation P.

Summary: Permutation Invariance and Equivariance

- **Permutation-invariant**

$$f(A, X) = f(PAP^T, PX)$$

Permute the input, the output stays the same. (map a graph to a vector)

- **Permutation-equivariant**

$$Pf(A, X) = f(PAP^T, PX)$$

Permute the input, output also permutes accordingly. (map a graph to a matrix)

Summary: Permutation Invariance and Equivariance

- **Permutation-invariant**

$$f(A, X) = f(PAP^T, PX)$$

Permute the input, the output stays the same. (map a graph to a vector)

- **Permutation-equivariant**

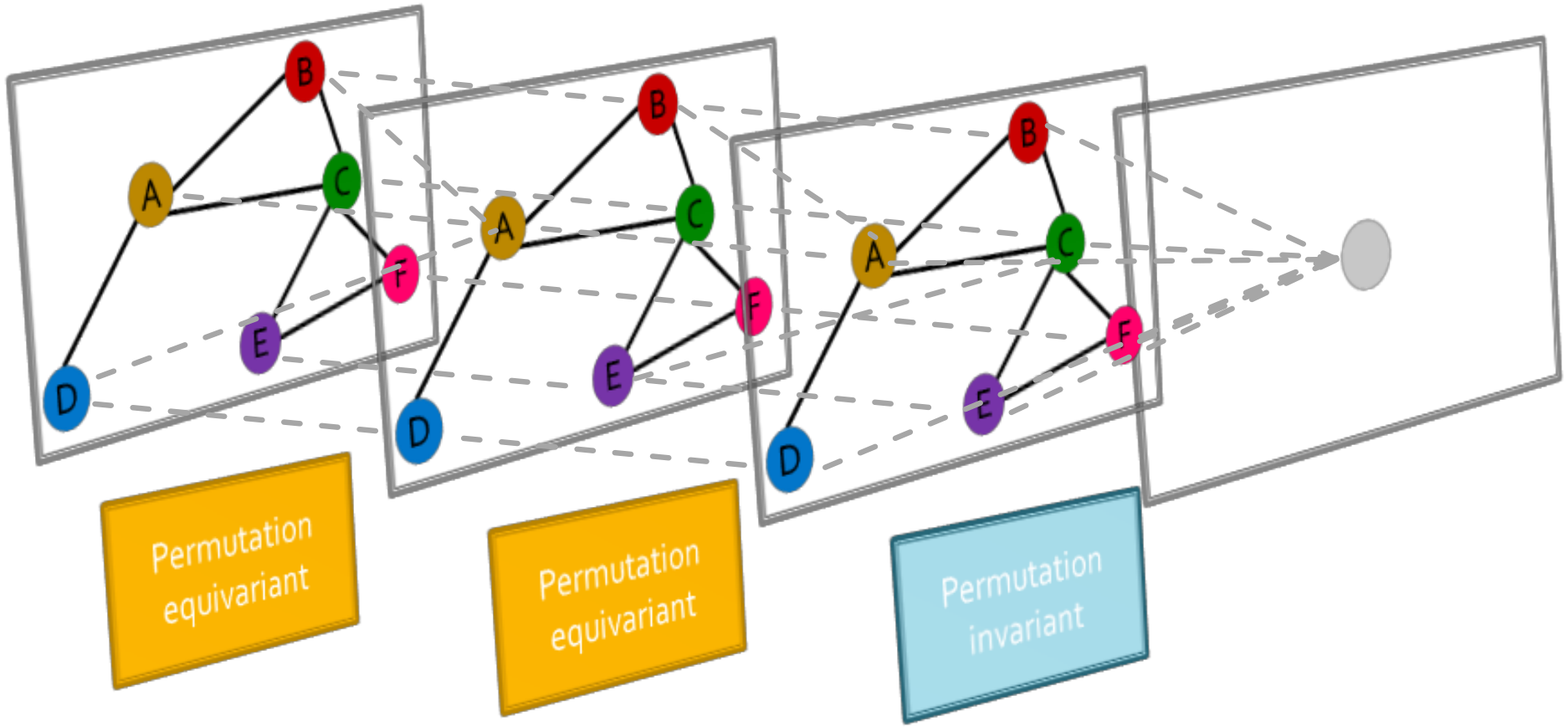
$$Pf(A, X) = f(PAP^T, PX)$$

Permute the input, output also permutes accordingly. (map a graph to a matrix)

- **Examples:**
  - $f(A, X) = 1^T X$: Permutation-**invariant**
    - Reason: $f(PAP^T, PX) = 1^T PX = 1^T X = f(A, X)$
  - $f(A, X) = X$: Permutation-**equivariant**
    - Reason: $f(PAP^T, PX) = PX = Pf(A, X)$
  - $f(A, X) = AX$: Permutation-**equivariant**
    - Reason: $f(PAP^T, PX) = PAP^T PX = PAX = Pf(A, X)$

Graph Neural Networks Overview

- GNNs consist of multiple permutation equivariant / invariant functions



[Bronstein, ICLR 2021 keynote]

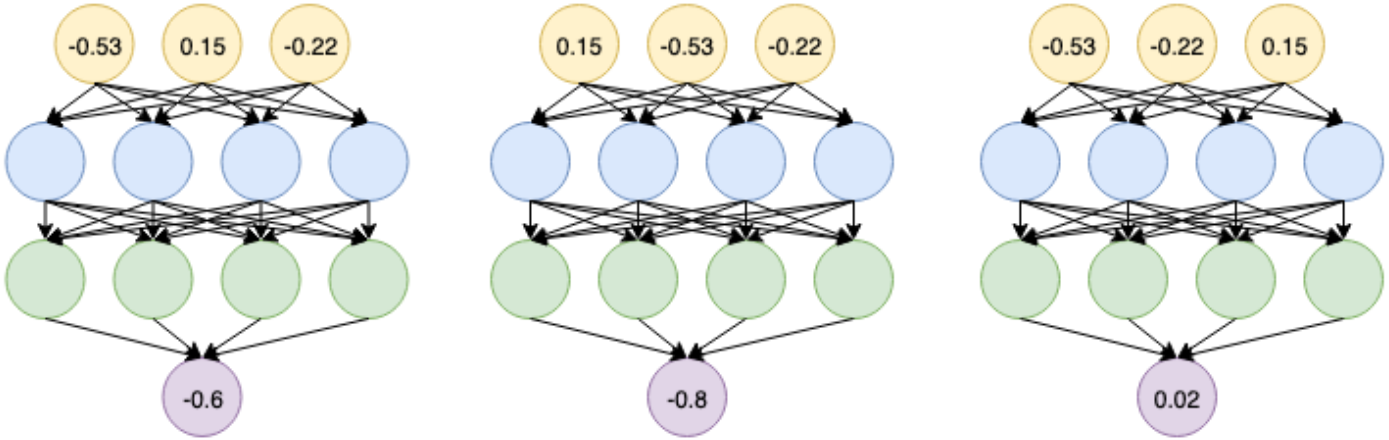
Graph Neural Networks Overview

- GNNs consist of multiple permutation equivariant / invariant functions

Are other neural network architectures, e.g., MLPs, permutation invariant / equivariant?

- **No.**

Switching the order of the input leads to different outputs!

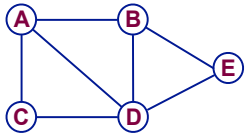


Graph Neural Networks Overview

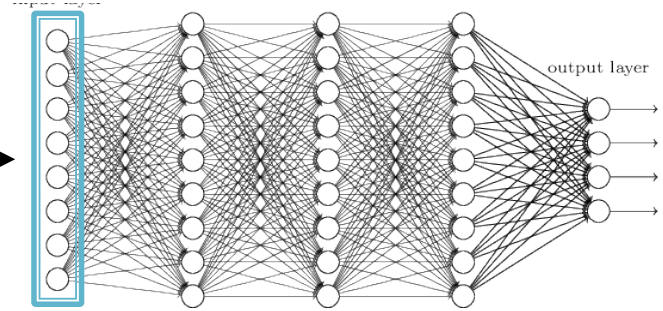
- GNNs consist of multiple permutation equivariant / invariant functions

Are other neural network architectures, e.g., MLPs, permutation invariant / equivariant?

- **No.**



	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



?

This explains why **the naïve MLP approach fails for graphs**!

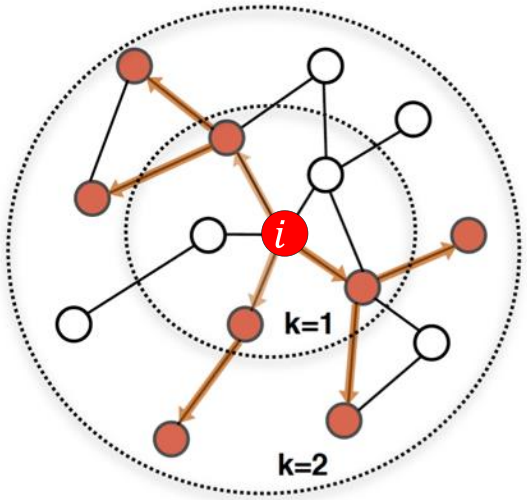
Graph Neural Networks Overview

- GNNs consist of multiple permutation equivariant / invariant functions

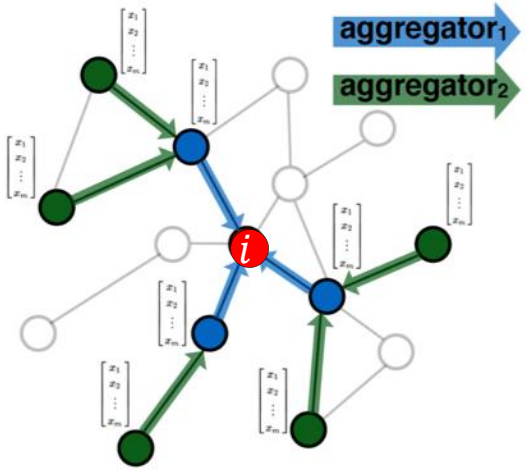
- Next: Design GNNs that are permutation equivariant / invariant by passing and aggregating information from neighbors

Graph Convolutional Networks

Idea: Node's neighborhood defines a computation graph



Determine node computation graph



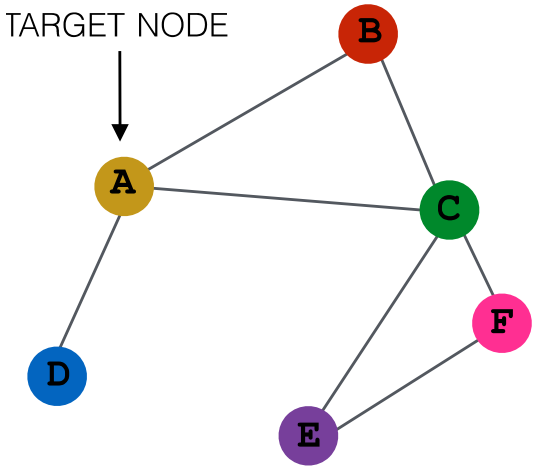
Propagate and transform information

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

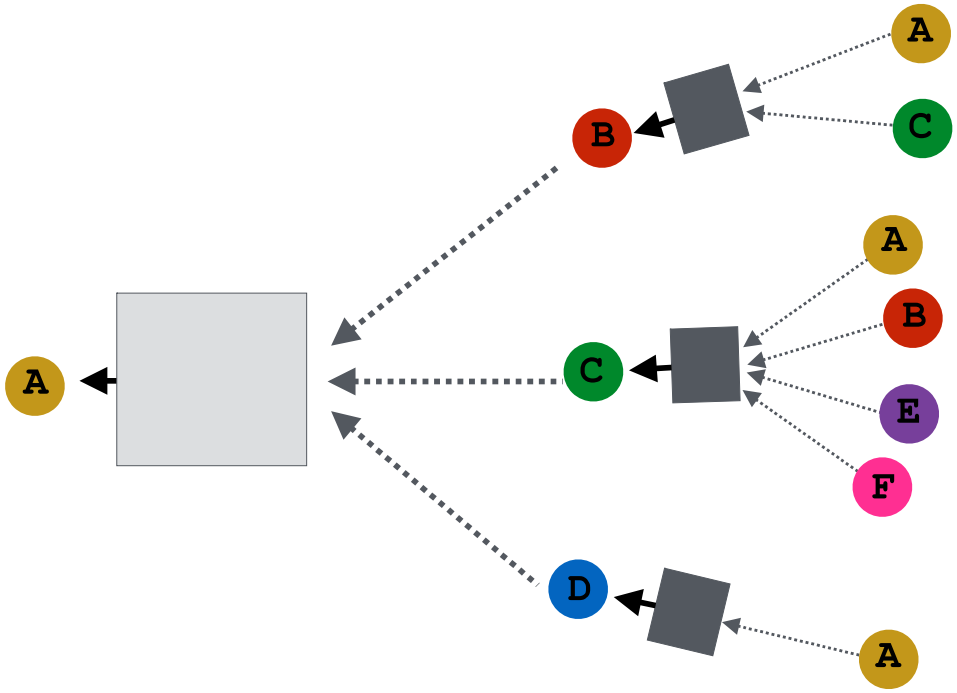
[Kipf and Welling, ICLR 2017]

Idea: Aggregate Neighbors

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

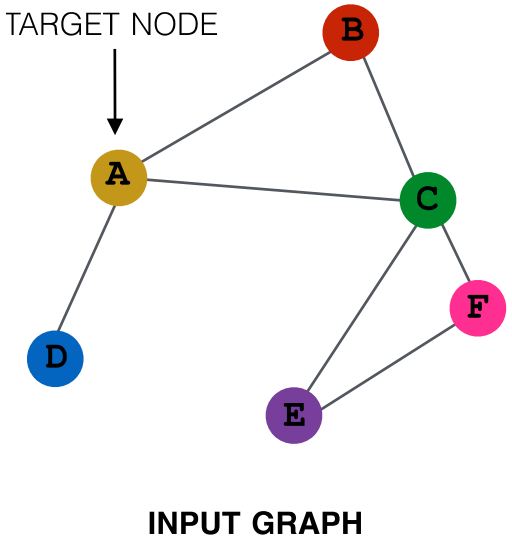


INPUT GRAPH

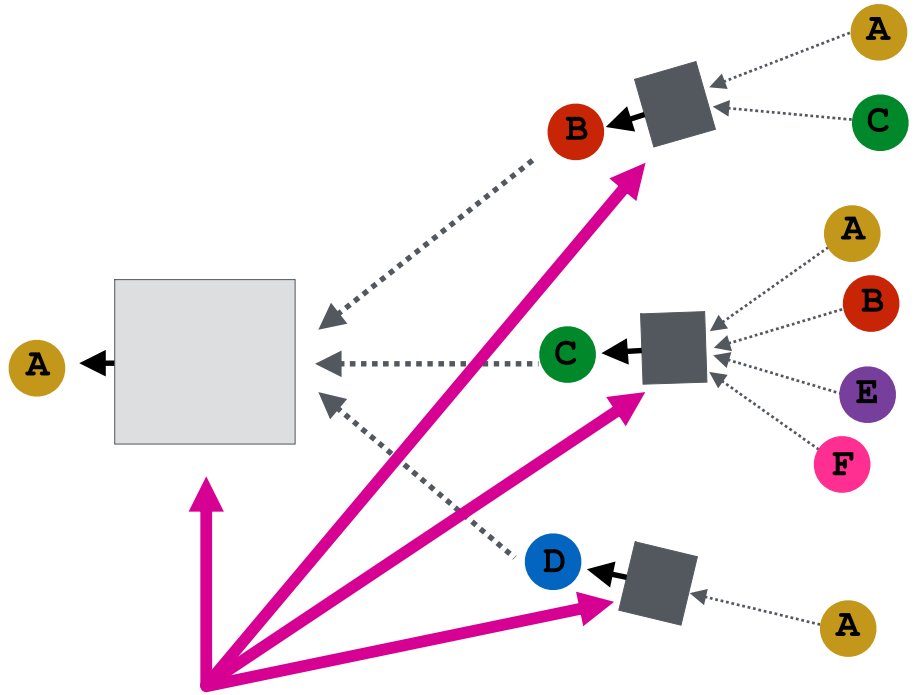


Idea: Aggregate Neighbors

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



INPUT GRAPH

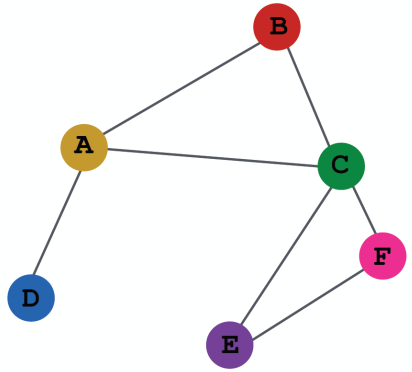


Neural networks

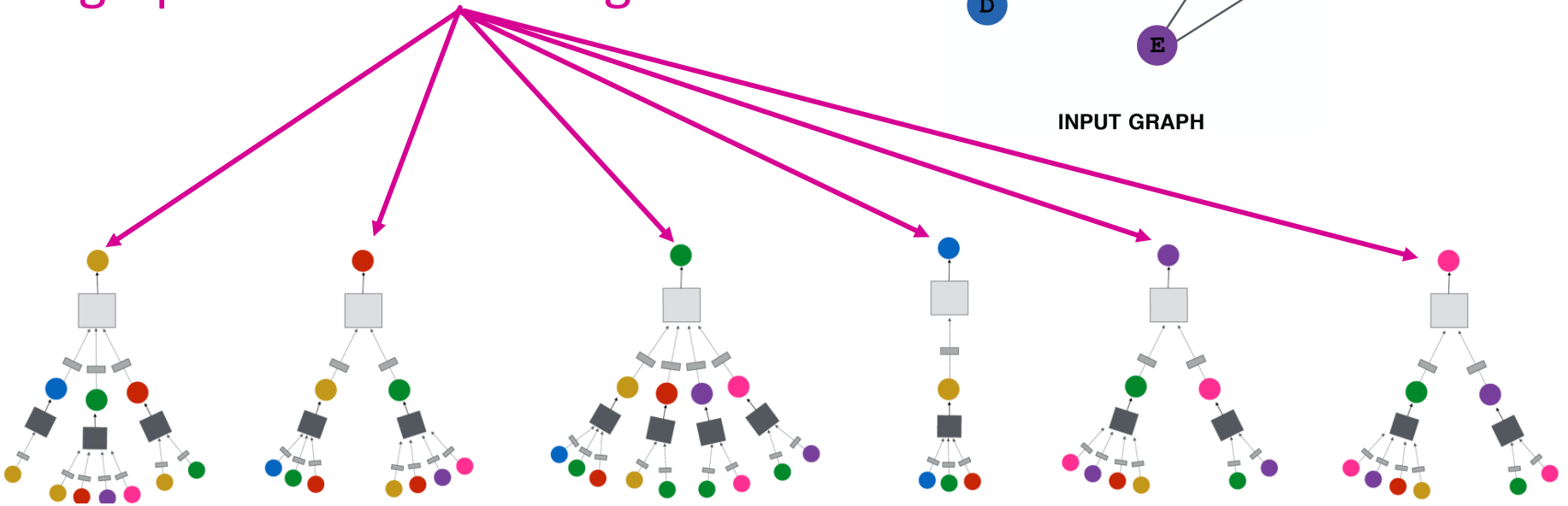
Idea: Aggregate Neighbors

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

Every node defines a computation graph based on its neighborhood!

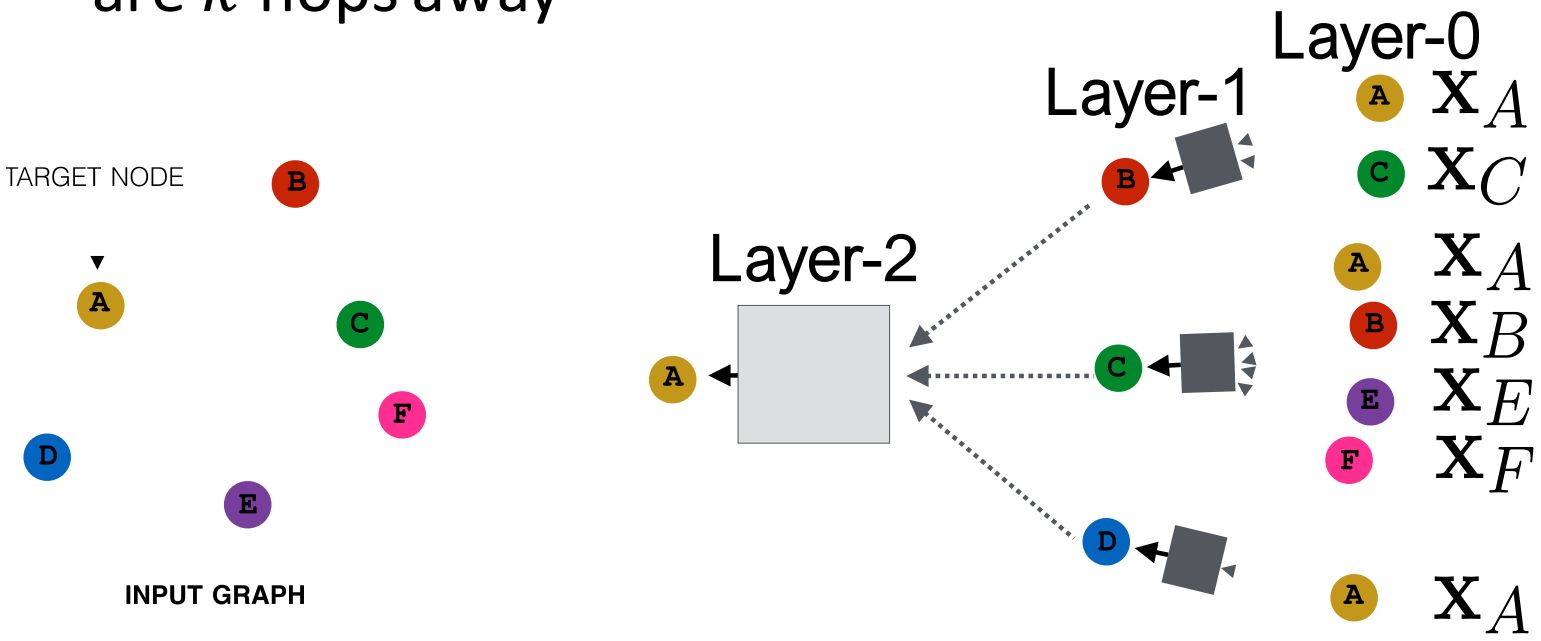


INPUT GRAPH



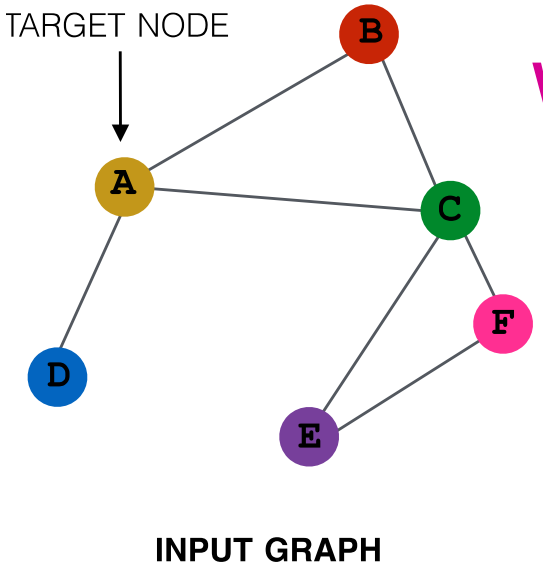
Deep Model: Many Layers

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

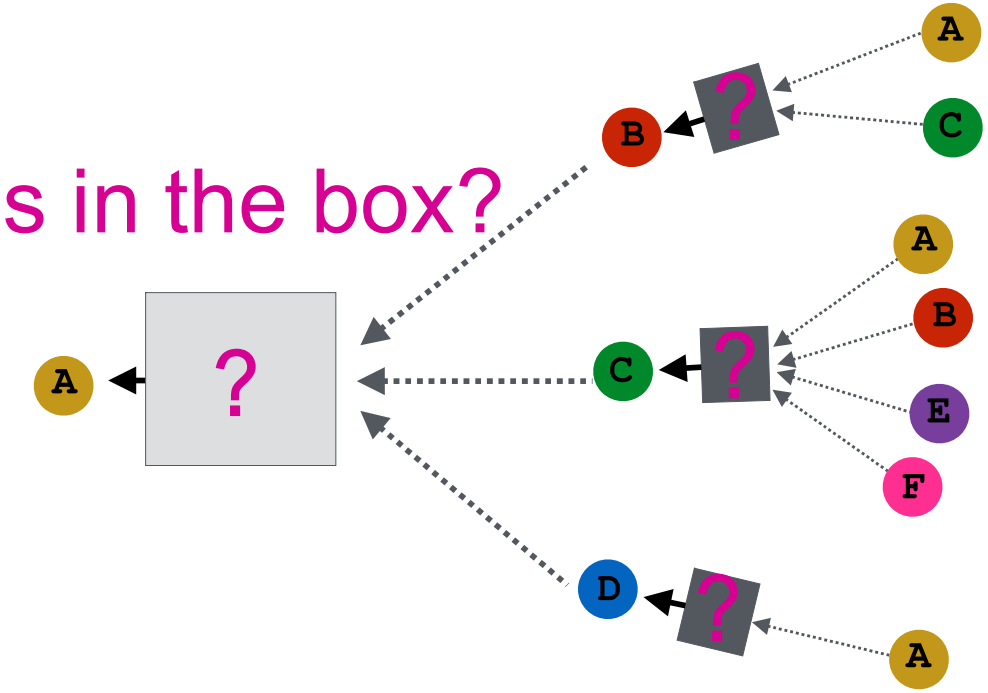


Neighborhood Aggregation

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

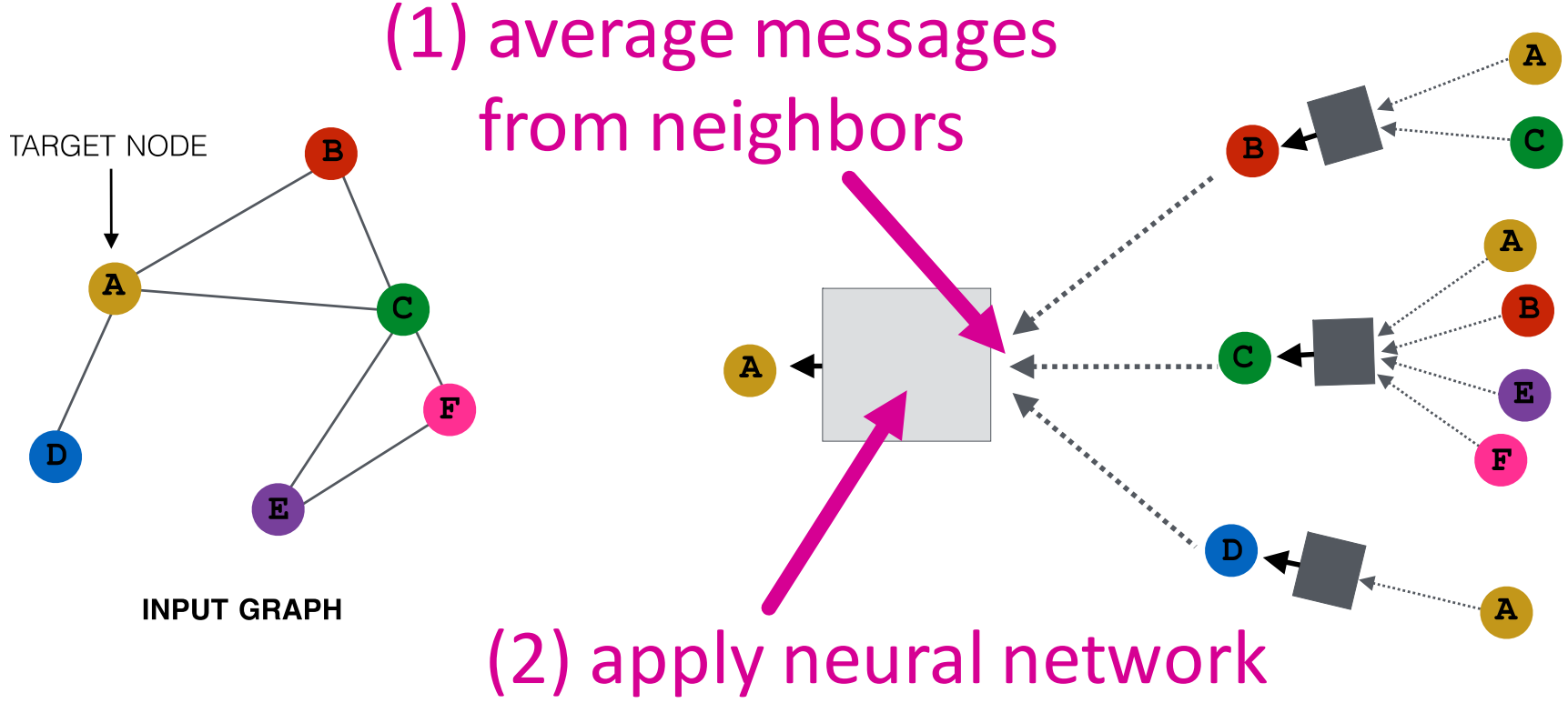


What is in the box?



Neighborhood Aggregation

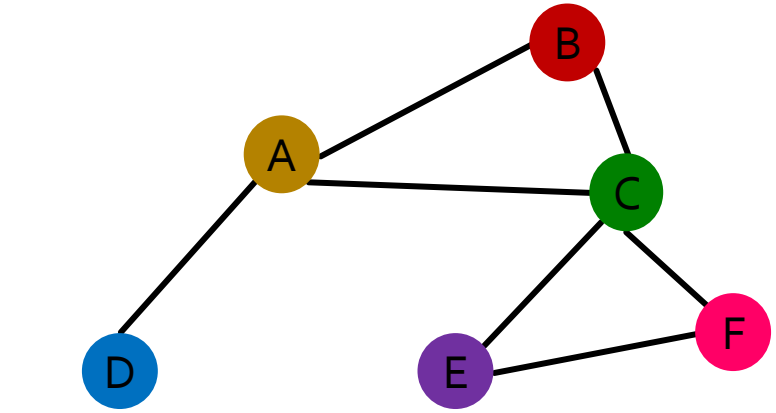
- **Basic approach:** Average information from neighbors and apply a neural network



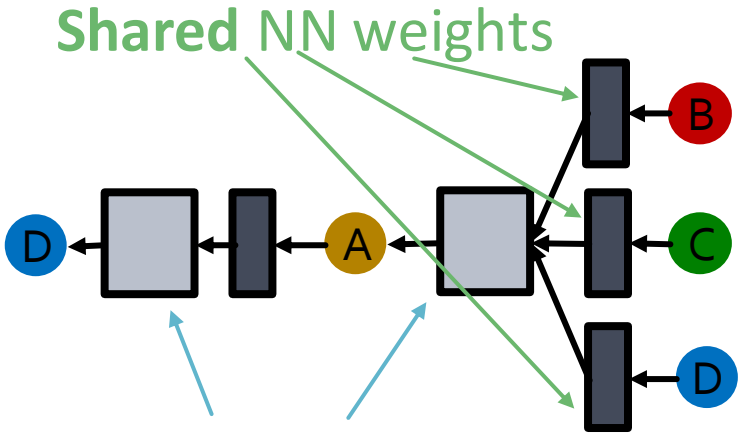
GCN: Invariance and Equivariance

What are the invariance and equivariance properties for a GCN?

- **Given a node**, the GCN that computes its embedding is **permutation invariant**



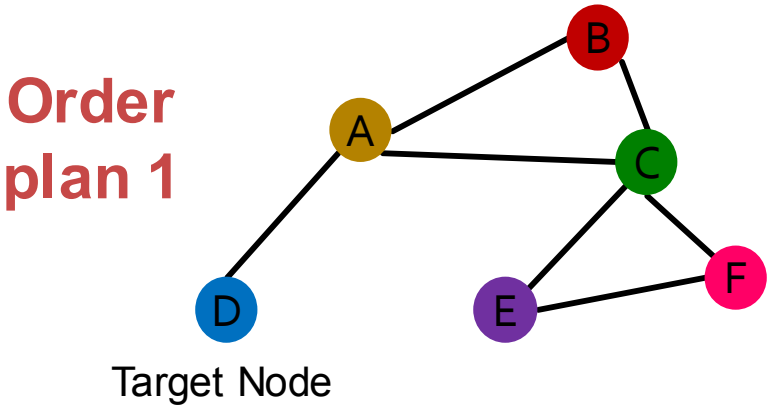
Target Node

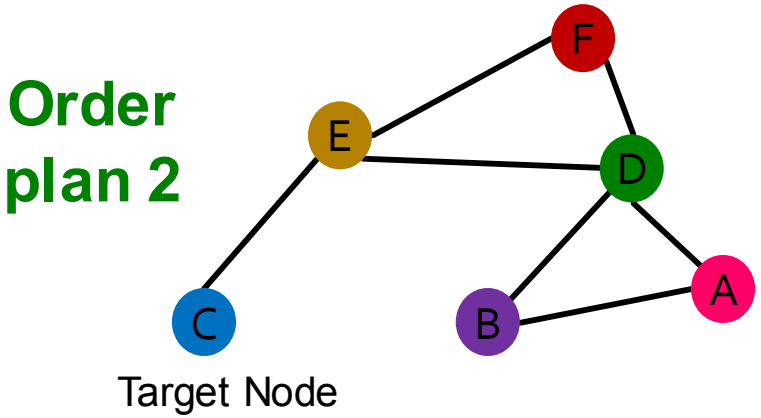


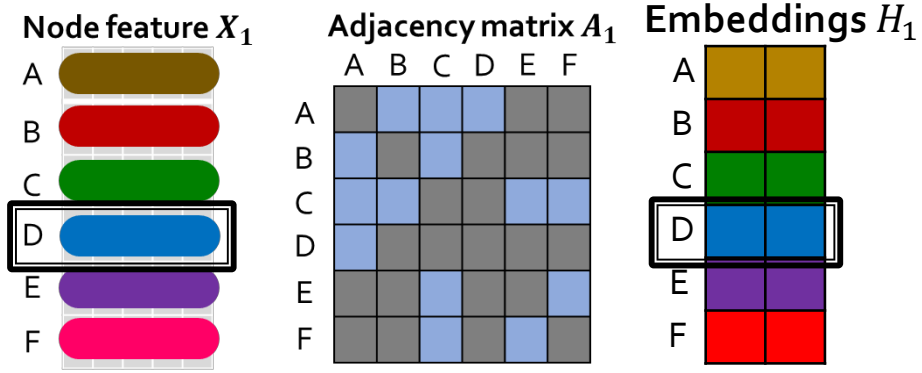
Average of neighbor's previous layer embeddings - **Permutation invariant**

GCN: Invariance and Equivariance

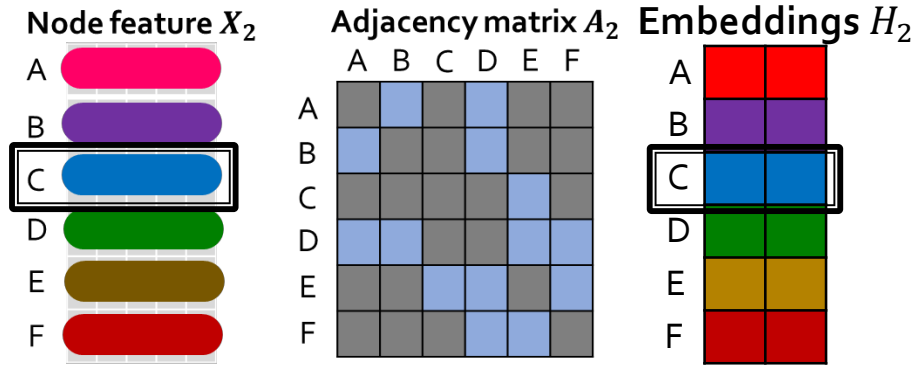
- **Considering all nodes in a graph**, GCN computation is **permutation equivariant**







Permute the input, the output also permutes accordingly - permutation equivariant



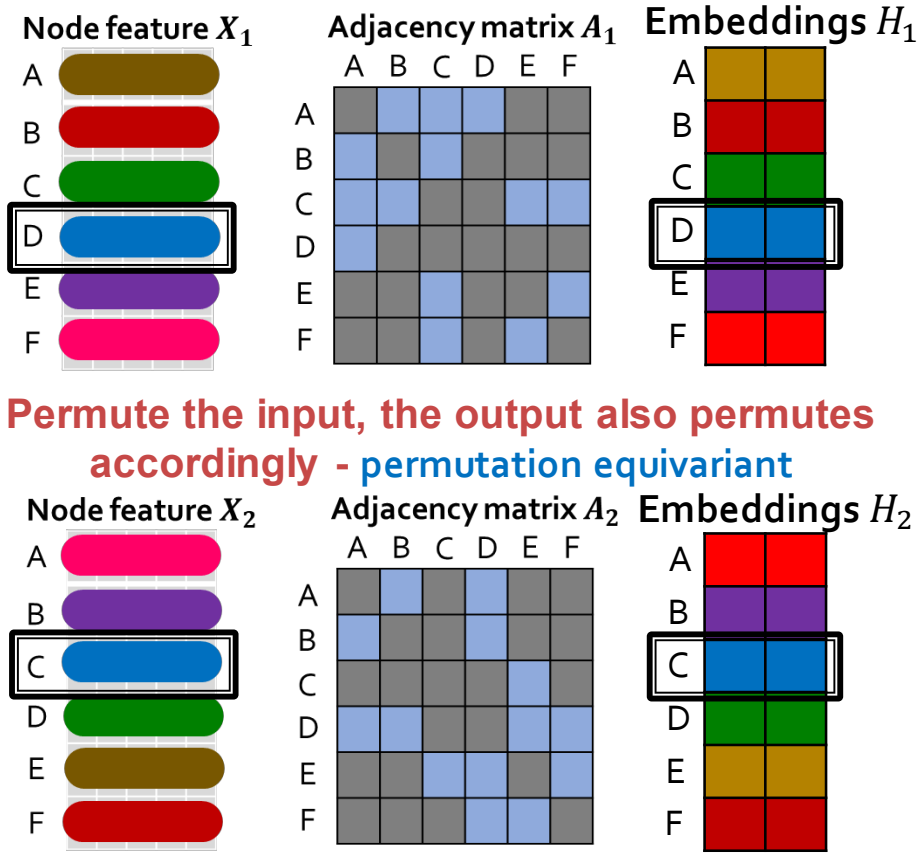
GCN: Invariance and Equivariance

- **Considering all nodes in a graph**, GCN computation is **permutation equivariant**

Detailed reasoning:

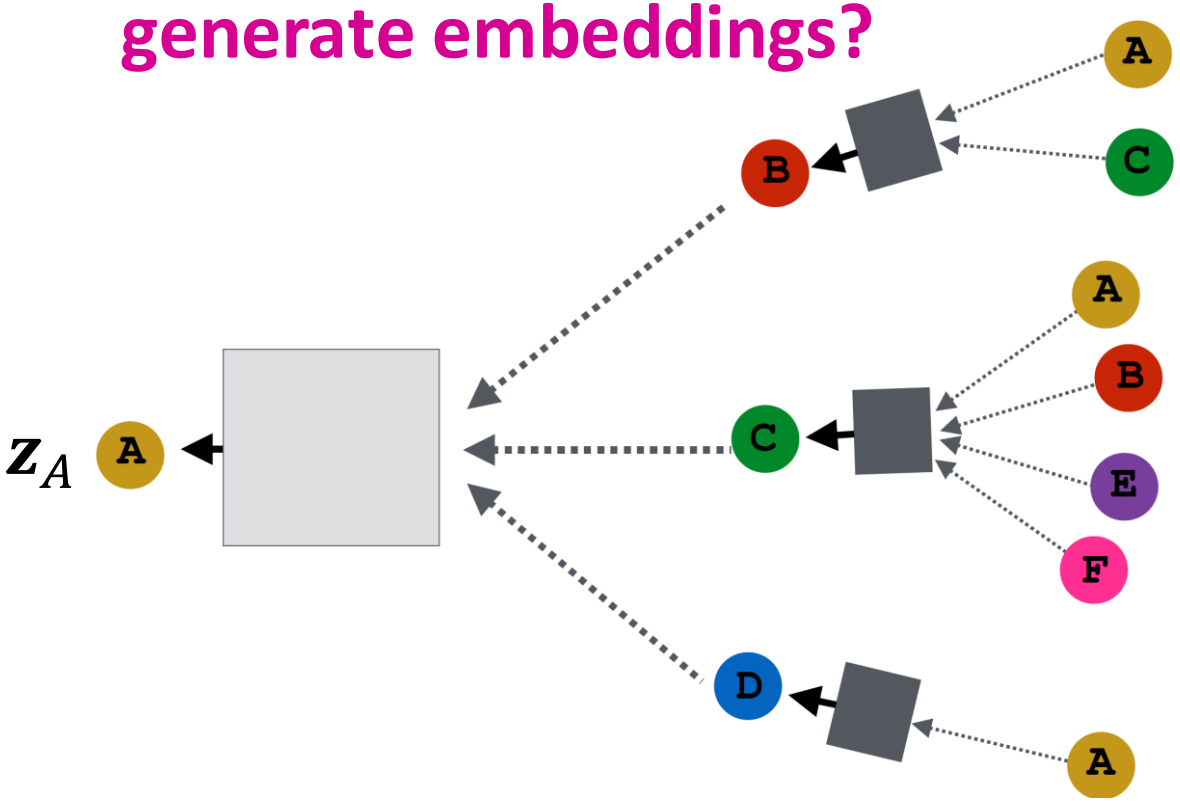
1. The rows of **input node features** and **output embeddings** are **aligned**
2. We know computing the embedding of **a given node** with GCN is **invariant.**
3. So, after permutation, the **location** of **a given node** in the **input node feature** matrix is changed, and the **the output embedding of a given node stays the same** (the colors of node feature and embedding are **matched**)

This is permutation equivariant



How to Train A GNN

How do we train the GCN to generate embeddings?



Need to define a loss function on the embeddings.

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
- **Unsupervised setting:**
  - No node label available
  - **Use the graph structure as the supervision!**

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):

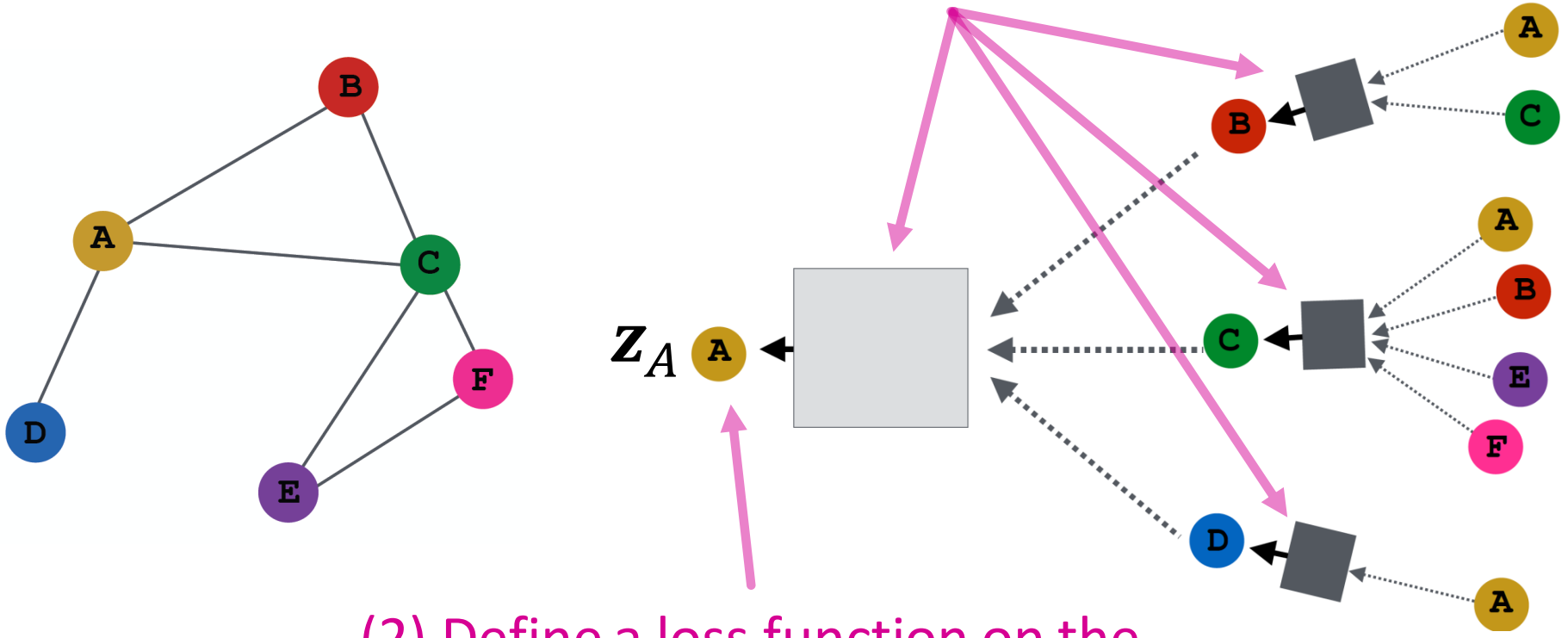
$$\min_{\Theta} \mathcal{L}(\boldsymbol{y}, f(\boldsymbol{z}_v))$$

  - $\boldsymbol{y}$: node label
  - $\mathcal{L}$ could be L2 if $\boldsymbol{y}$ is real number, or cross entropy if $\boldsymbol{y}$ is categorical
- **Unsupervised setting:**
  - No node label available
  - **Use the graph structure as the supervision!**

"Similar" nodes have similar embeddings (discussed in last lecture)

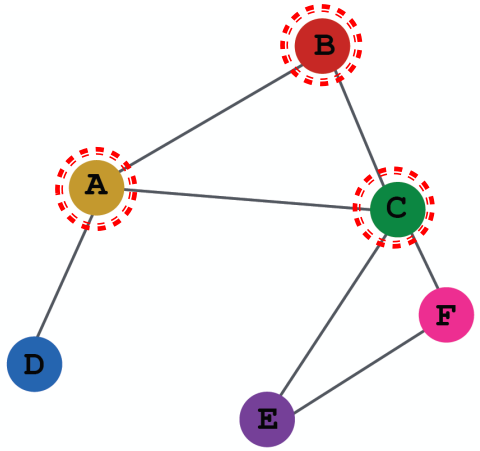
Model Design: Overview

(1) Define a neighborhood aggregation function



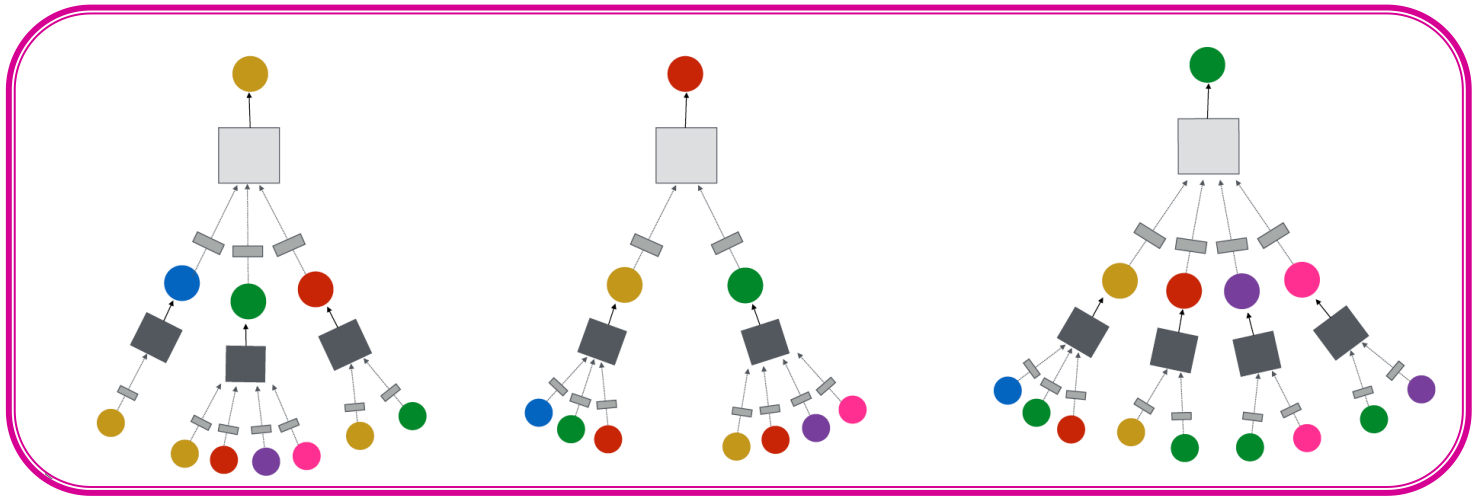
(2) Define a loss function on the embeddings

Model Design: Overview

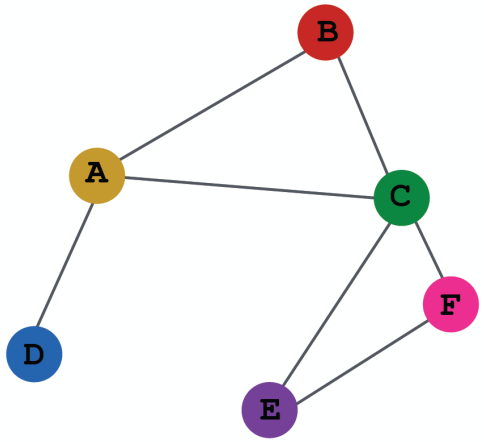


INPUT GRAPH

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



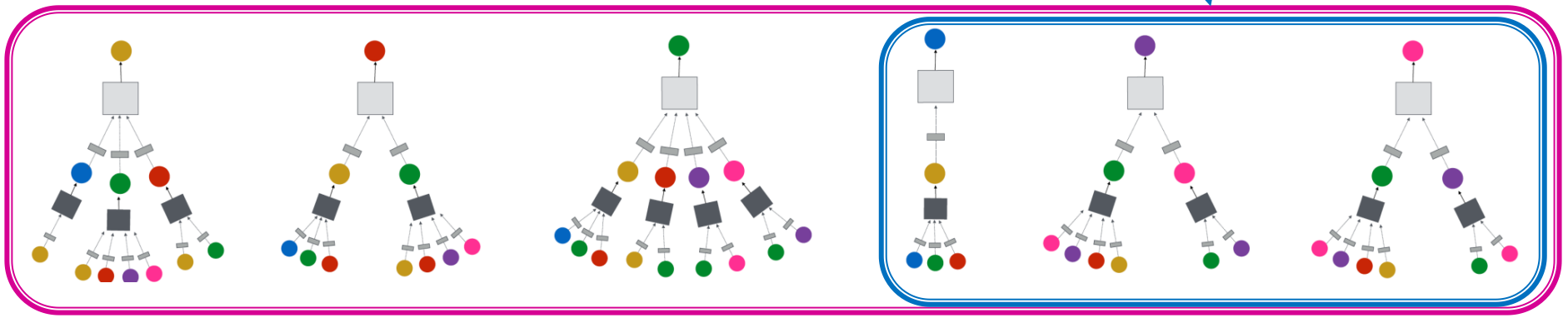
Model Design: Overview



INPUT GRAPH

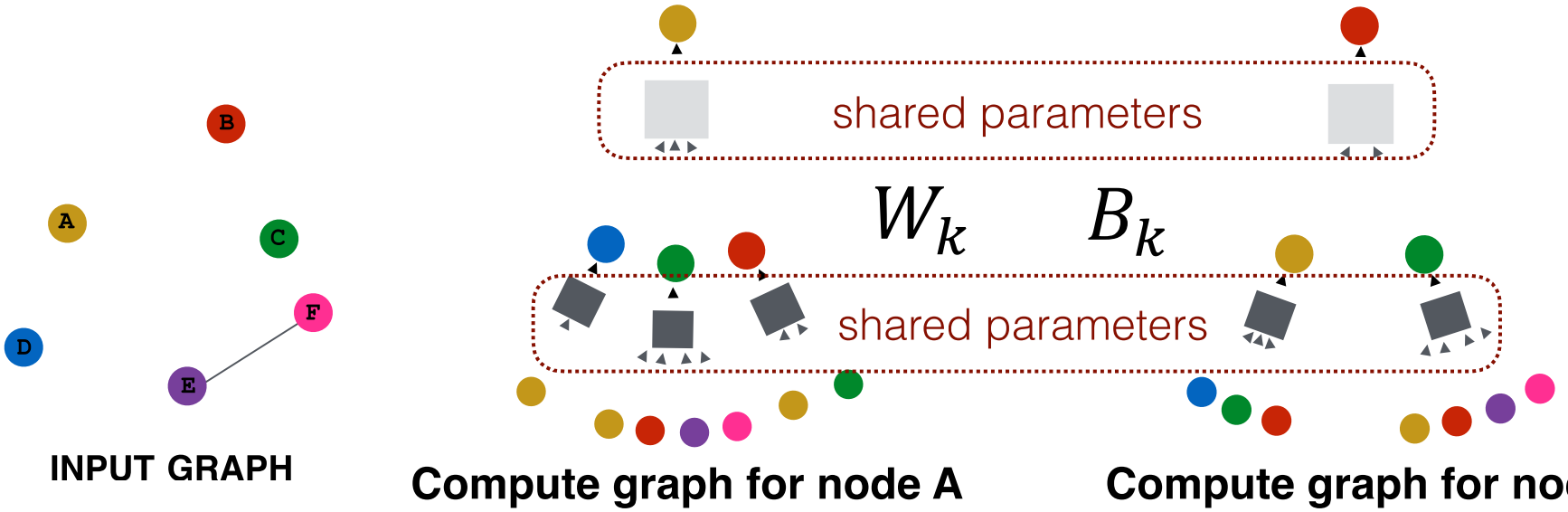
(4) Generate embeddings for nodes as needed

Even for nodes we never trained on!

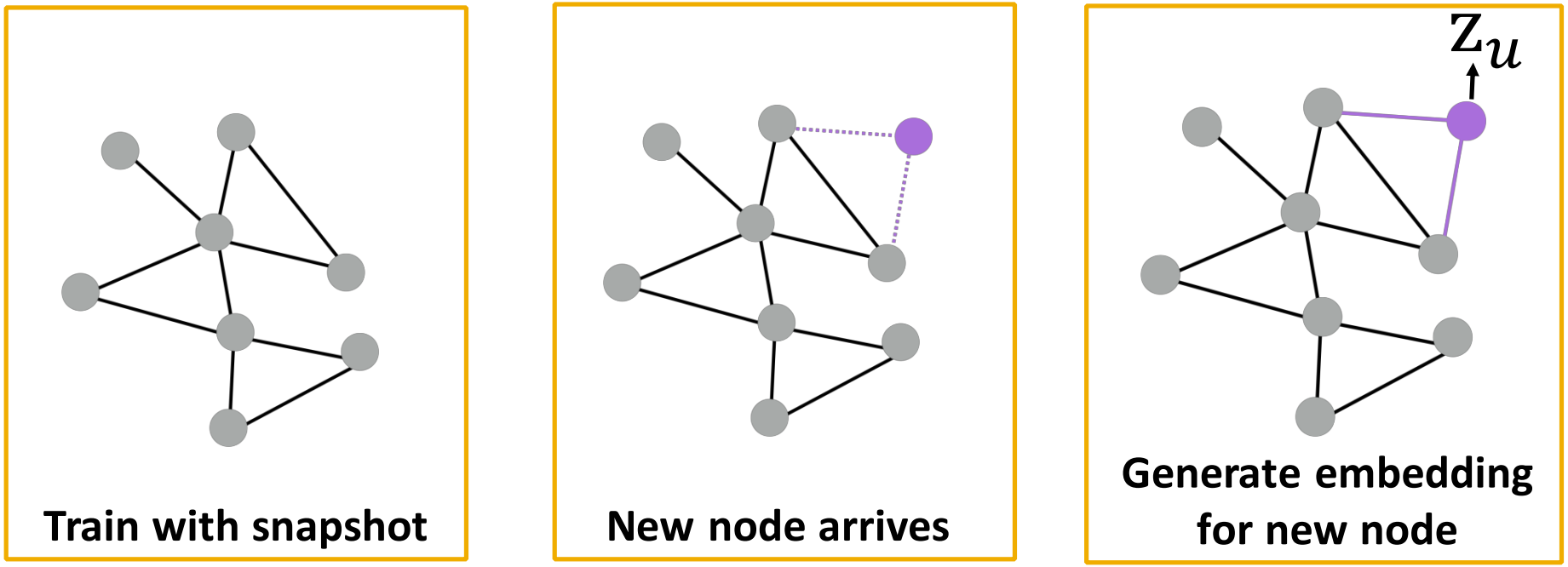


Inductive Capability

- **The same aggregation parameters are shared for all nodes:**
  - The number of model parameters is sublinear in $|V|$ and we can **generalize to unseen nodes**!

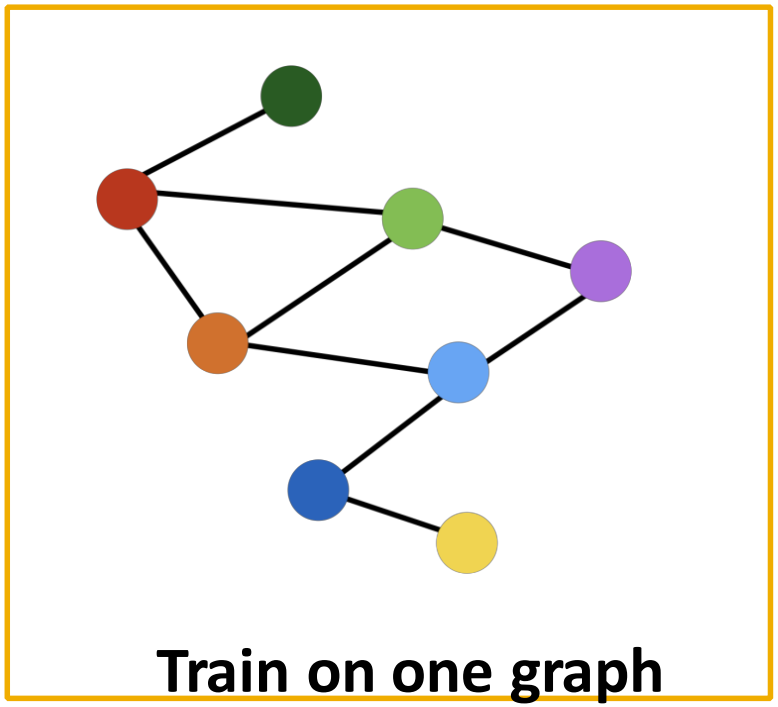


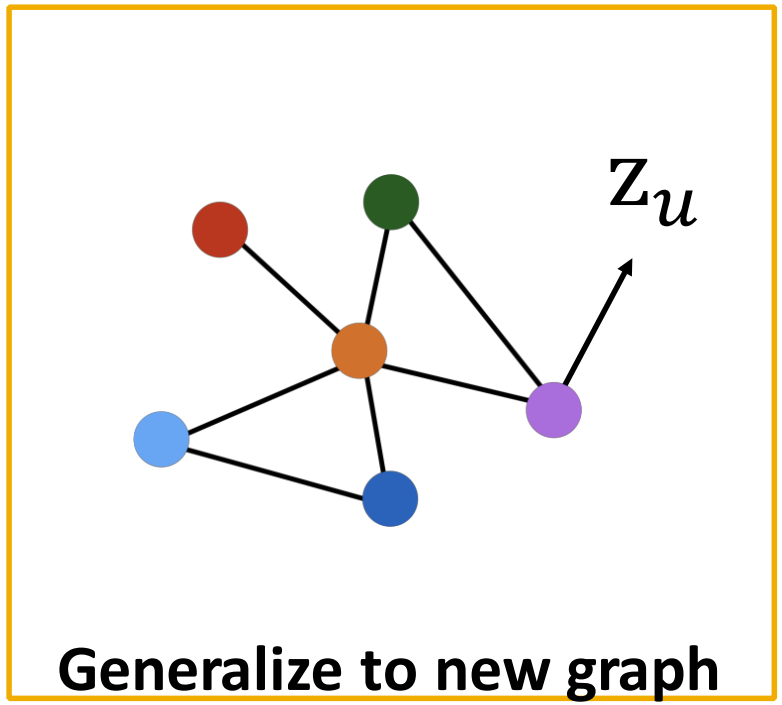
Inductive Capability: New Nodes



- Many application settings constantly encounter previously unseen nodes:
  - E.g., Reddit, YouTube, Google Scholar
- Need to generate new embeddings "on the fly"

Inductive Capability: New Graphs

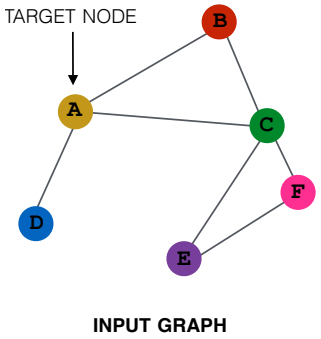


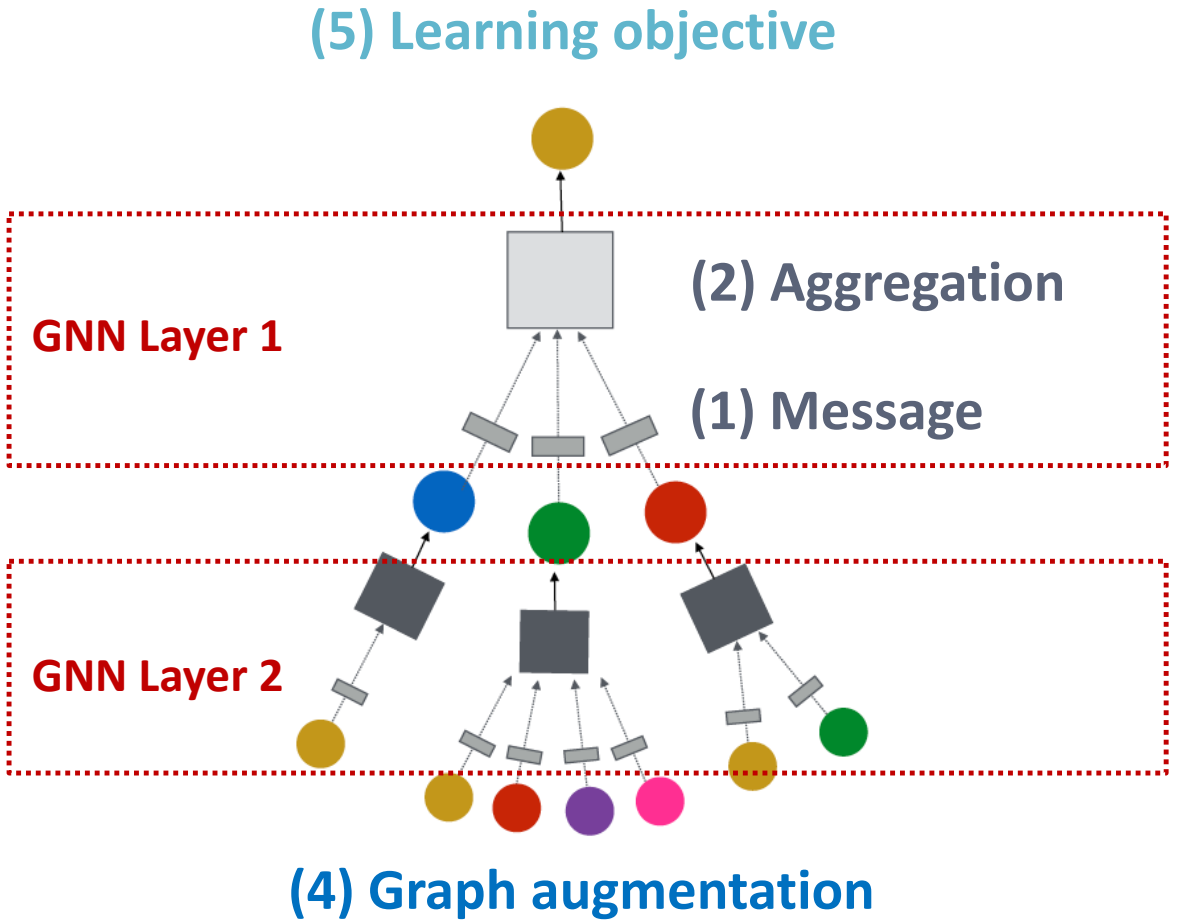


Inductive node embedding → Generalize to entirely unseen graphs

E.g., train on protein interaction graph from model organism A and generate embeddings on newly collected data about organism B

Discussion: Design Space of GNNs





J. You, R. Ying, J. Leskovec. Design Space of Graph Neural Networks, NeurIPS 2020

Ex1: Connectivity

Our assumption so far has been

- **Raw input graph = computational graph**

Reasons for breaking this assumption

- **Feature level:**
  - The input graph **lacks features** → feature augmentation
- **Structure level:**
  - The graph is **too sparse** → inefficient message passing
  - The graph is **too dense** → message passing is too costly
  - The graph is **too large** → cannot fit the computational graph into a GPU
- It's just **unlikely that the input graph happens to be the optimal computation graph** for embeddings

Ex1: Connectivity

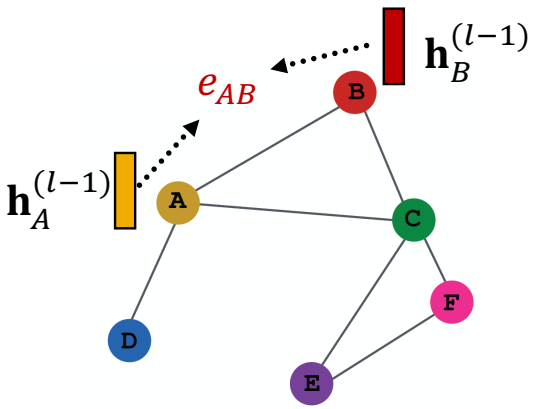
- **Graph Feature manipulation**
  - The input graph **lacks features → feature augmentation**
- **Graph Structure manipulation**
  - The graph is **too sparse → Add virtual nodes / edges**
  - The graph is **too dense → Sample neighbors when doing message passing**
  - The graph is **too large → Sample subgraphs to compute embeddings**
    - Will cover later in lecture: Scaling up GNNs

Ex2: Graph Attention Network (GAT)

- **In GCN**
  - $\alpha_{vu} = \frac{1}{|N(v)|}$ is the **weighting factor (importance)** of node u's message to node v
  - $\Longrightarrow \alpha_{vu}$ is defined **explicitly** based on the structural properties of the graph (node degree)
  - $\Longrightarrow$ All neighbors $u \in N(v)$ are equally important to node v

Not all node's neighbors are equally important

- Query, Key, Value
- Alignment $\boldsymbol{e}$
- $\boldsymbol{a} = \text{softmax}(\boldsymbol{e})$



Questions?