DSC250: Advanced Data Mining

Graph Mining

Zhiting Hu

Lecture 11, Feb 11, 2025

UC San Diego
HALICIOĞLU DATA SCIENCE INSTITUTE

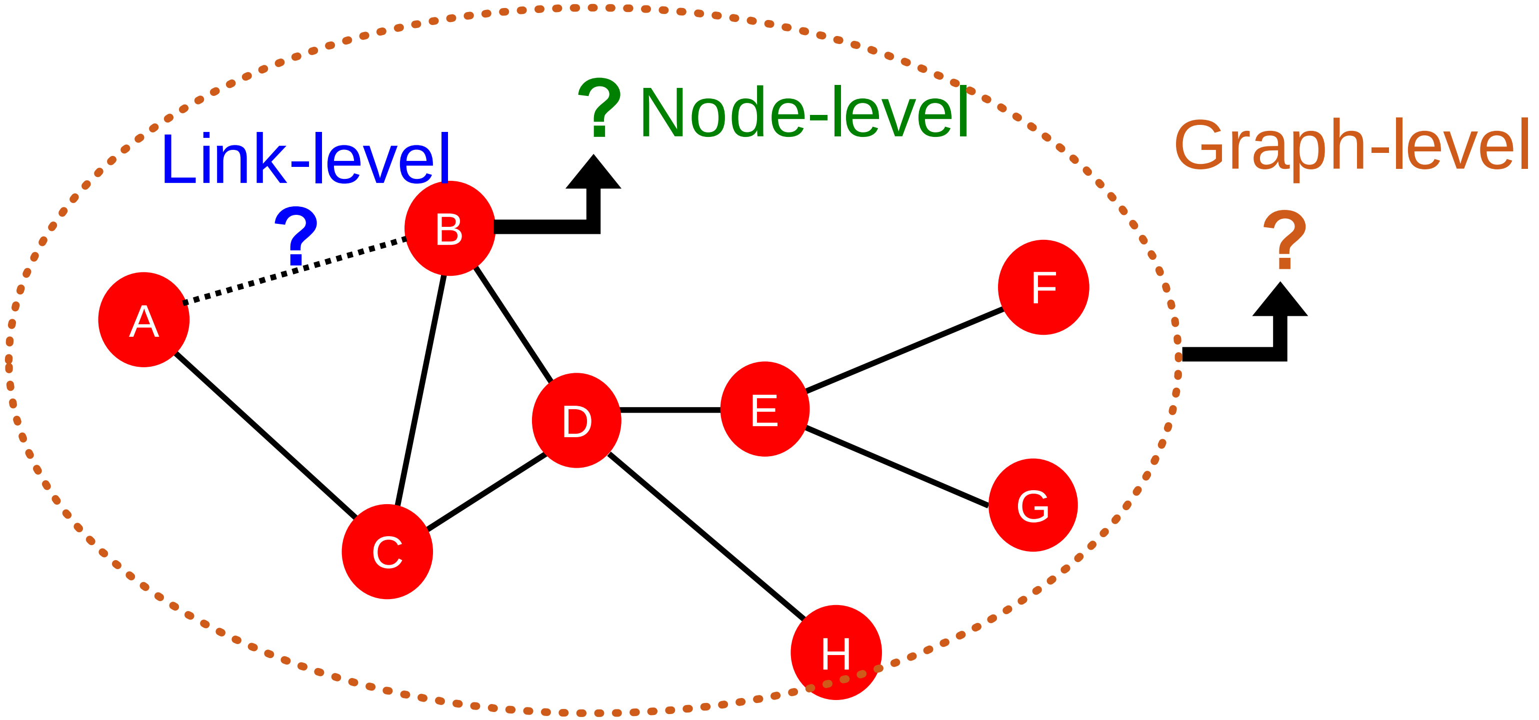
Outline

- Graph representation learning

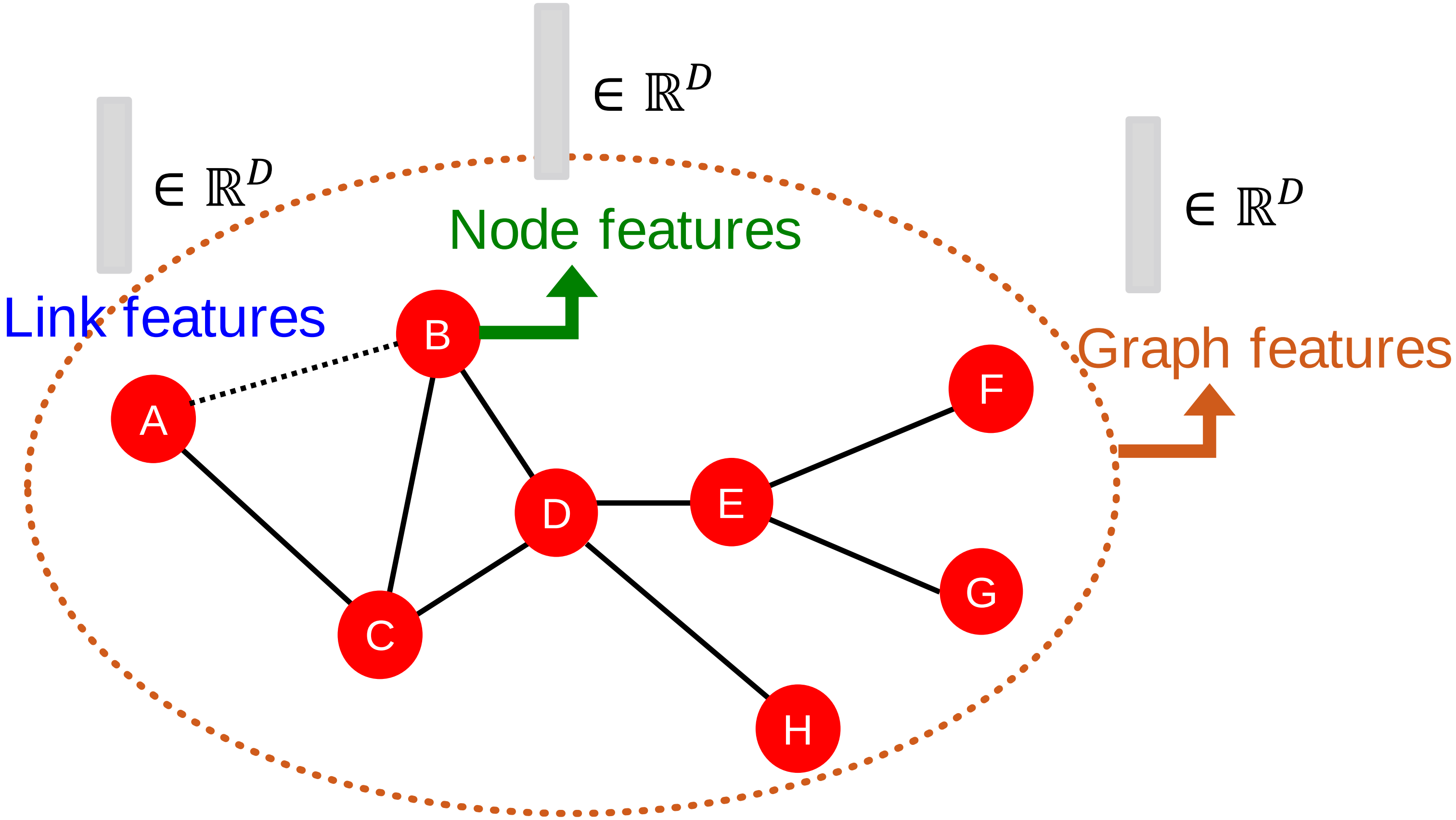
- Presentation
  - Tianhao Zhou, Hao Wang: "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks”
  - Nikhil Chowdary Paleti, Shankara Narayanan Venkateswara Raju: “DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning”
  - Letong Liang, Selena Ge: “s1: Simple test-time scaling”
  - Jahnavi Patel, Pratishtha Gaur: “From Local to Global: A Graph RAG Approach to Query-Focused Summarization”

Recap: Tasks on Graph

- Node-level prediction
- Link-level prediction
- Graph-level prediction

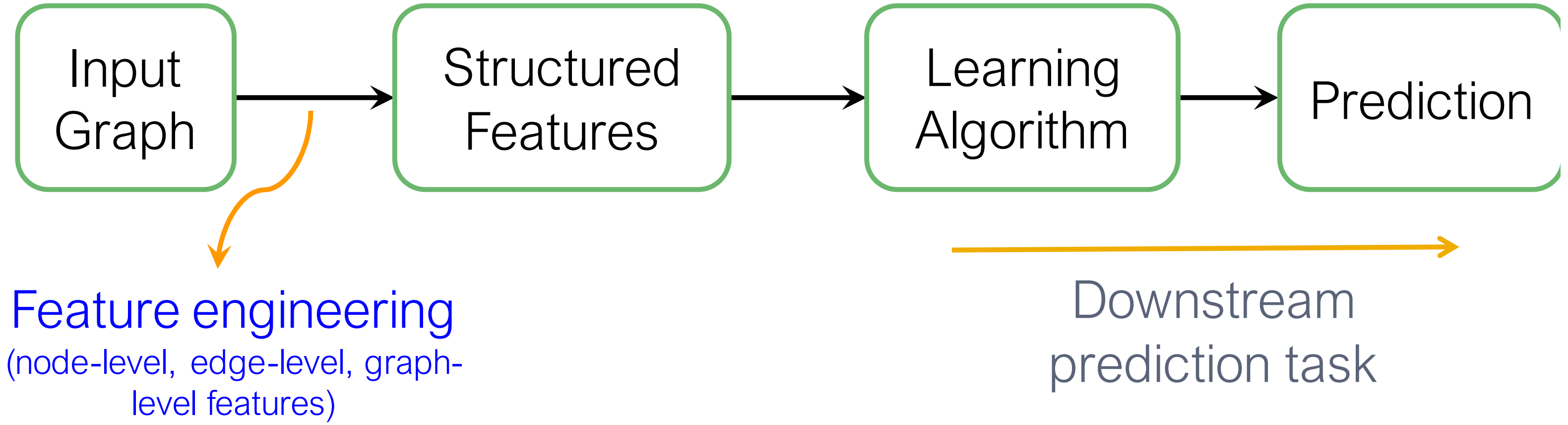


Recap: Getting Features for Nodes/Links/Graphs



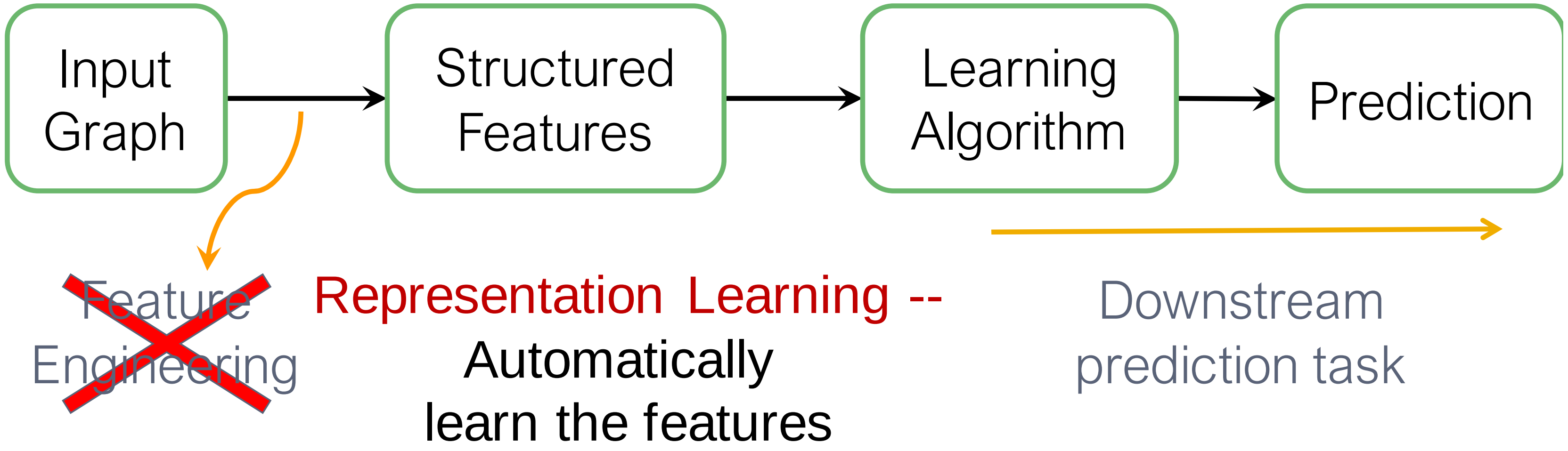
Recap: feature engineering

- Node-level:
  - Node degree, centrality, clustering coefficient, graphlets
- Link-level:
  - Distance-based feature
  - Local/global neighborhood overlap
- Graph-level:
  - Graphlet kernel



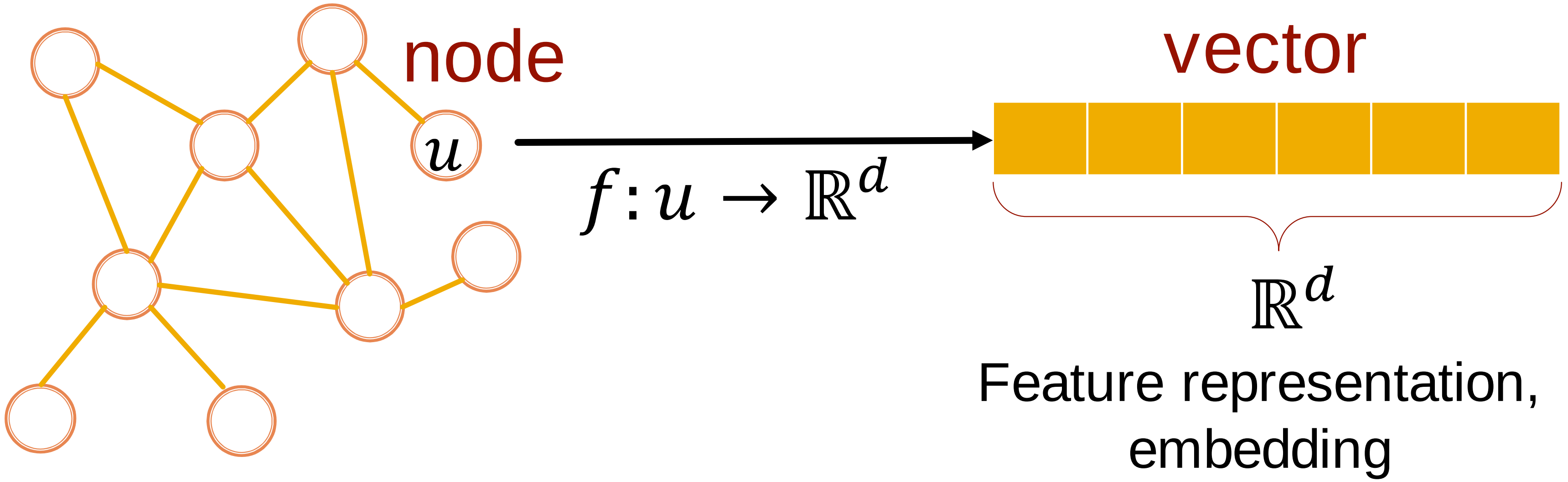
Graph Representation Learning

Graph Representation Learning alleviates the need to do feature engineering every single time.



Graph Representation Learning

Goal: Efficient task-independent feature learning for machine learning with graphs!



Node Embedding

- **Task: Map nodes into an embedding space**
  - Similarity of embeddings between nodes indicates their similarity in the network. For example:
    - Both nodes are close to each other (connected by an edge)
  - Encode network information
  - Potentially used for many downstream predictions

Vec

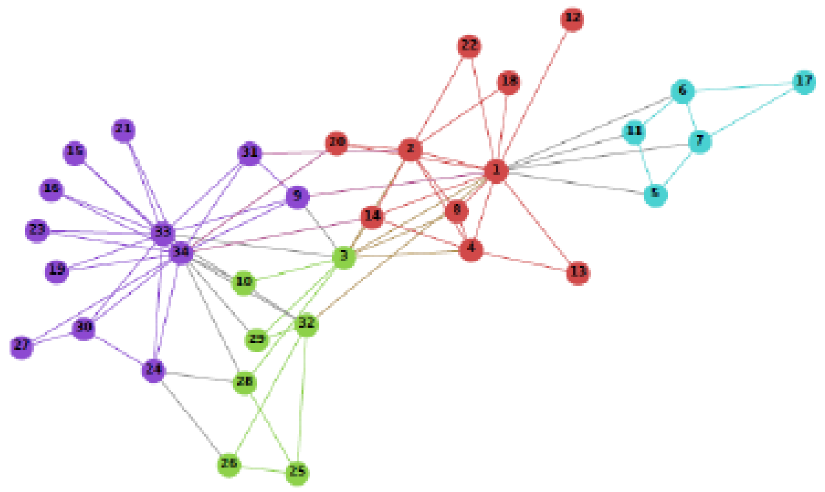
embeddings $\mathbb{R}^d$

Tasks

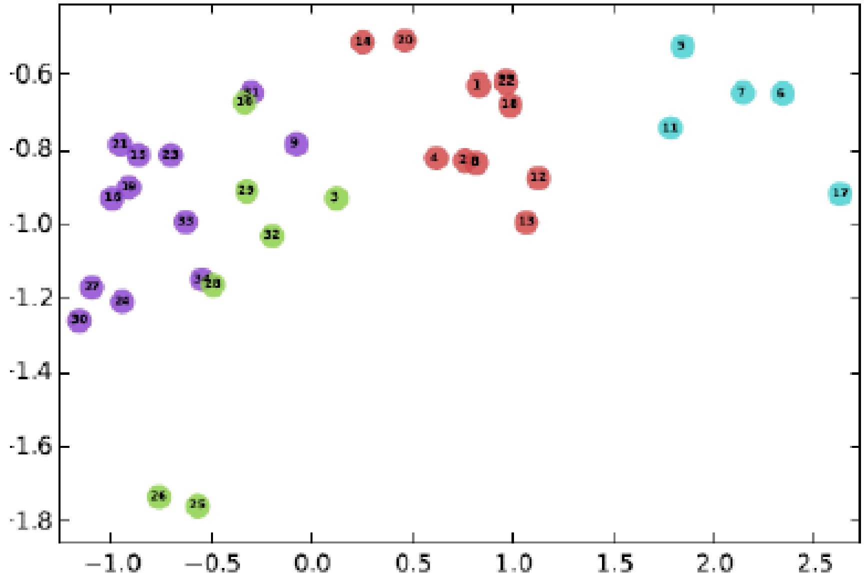
- Node classification
- Link prediction
- Graph classification
- Anomalous node detection
- Clustering
- ….

Example Node Embedding

- **2D embedding of nodes of the Zachary's Karate Club network:**



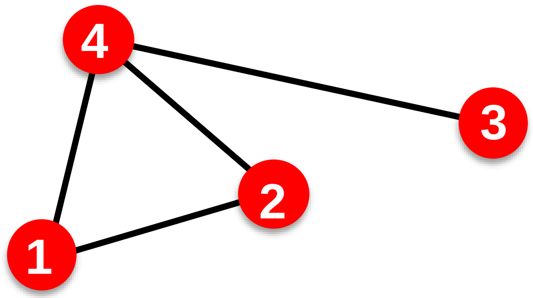
Input



Output

Node Embedding: Setup

- **Assume we have a graph G:**
  - V is the vertex set.
  - $\mathbf{A}$ is the adjacency matrix (assume binary).
  - **For simplicity: No node features or extra information is used**

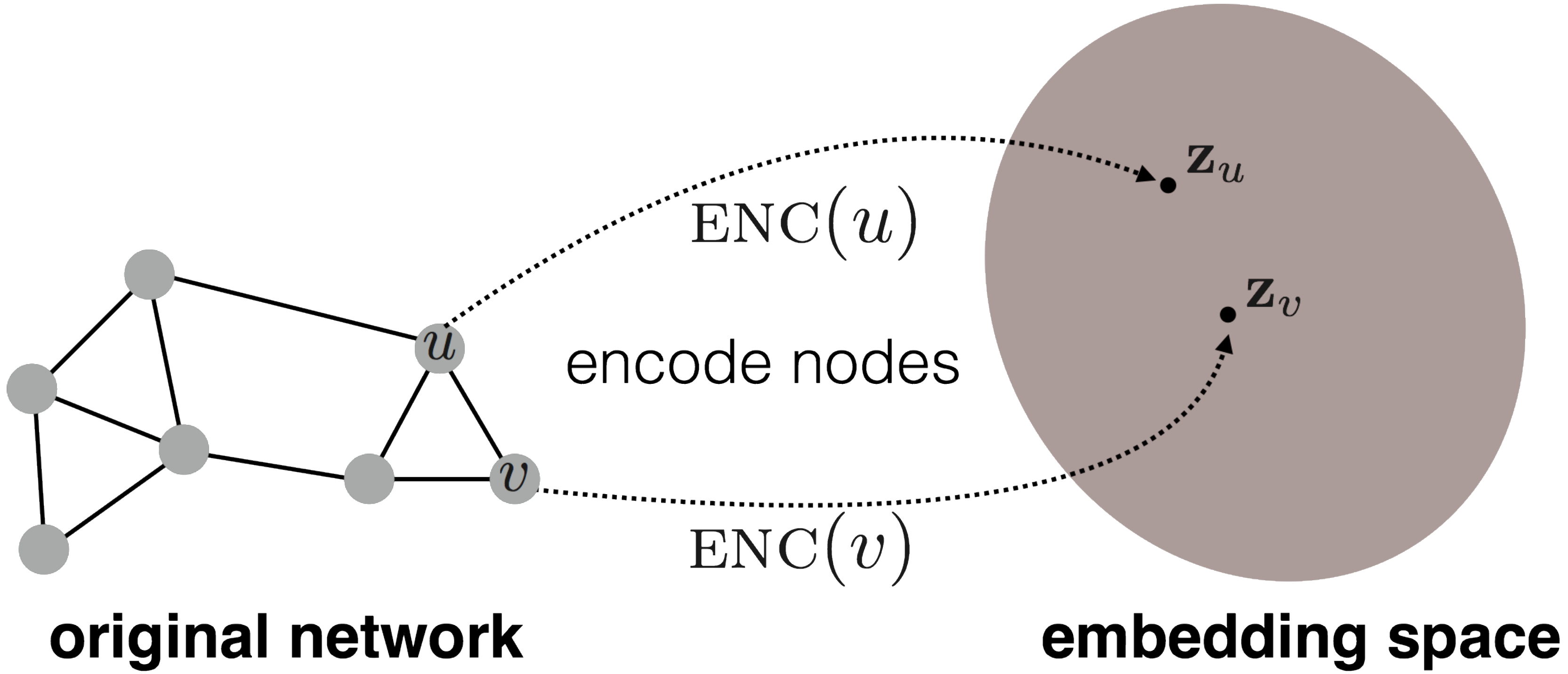


V: {1, 2, 3, 4}

$$A = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$

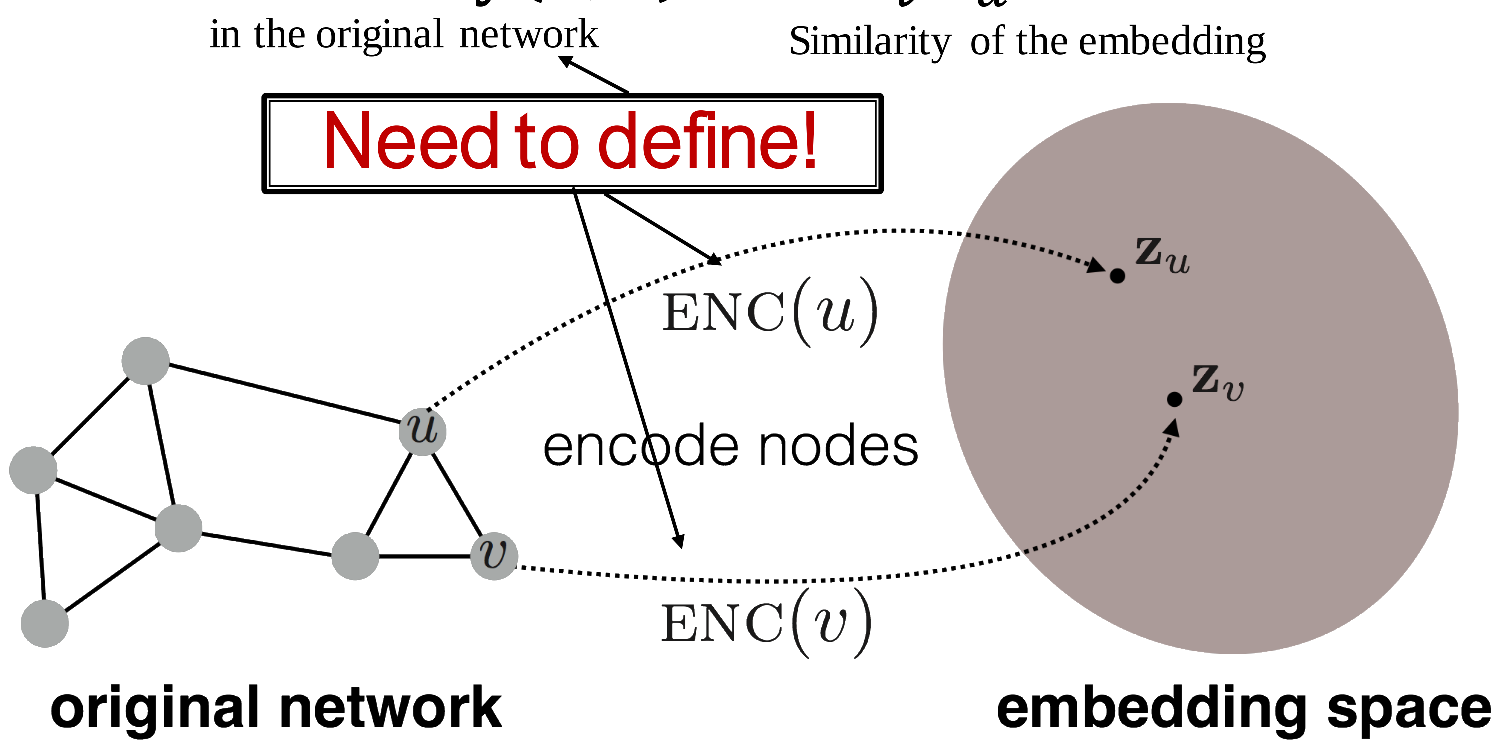
Node Embedding

- Goal is to encode nodes so that similarity in the embedding space (e.g., dot product) approximates similarity in the graph



Node Embedding

Goal: $\text{similarity}(u, v) \approx \mathbf{z}_v^{\mathrm{T}} \mathbf{z}_u$



Node Embedding: Key Components

- **Encoder:** maps each node to a low-dimensional vector

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

d-dimensional embedding

node in the input graph

- **Similarity function:** specifies how the relationships in vector space map to the relationships in the original network

$$\text{similarity}(u, v) \approx \mathbf{z}_v^{\mathrm{T}} \mathbf{z}_u$$

Decoder

Similarity of u and v in the original network

dot product between node embeddings

"Shallow" Encoding

Simplest encoding approach: **Encoder is just an embedding-lookup**

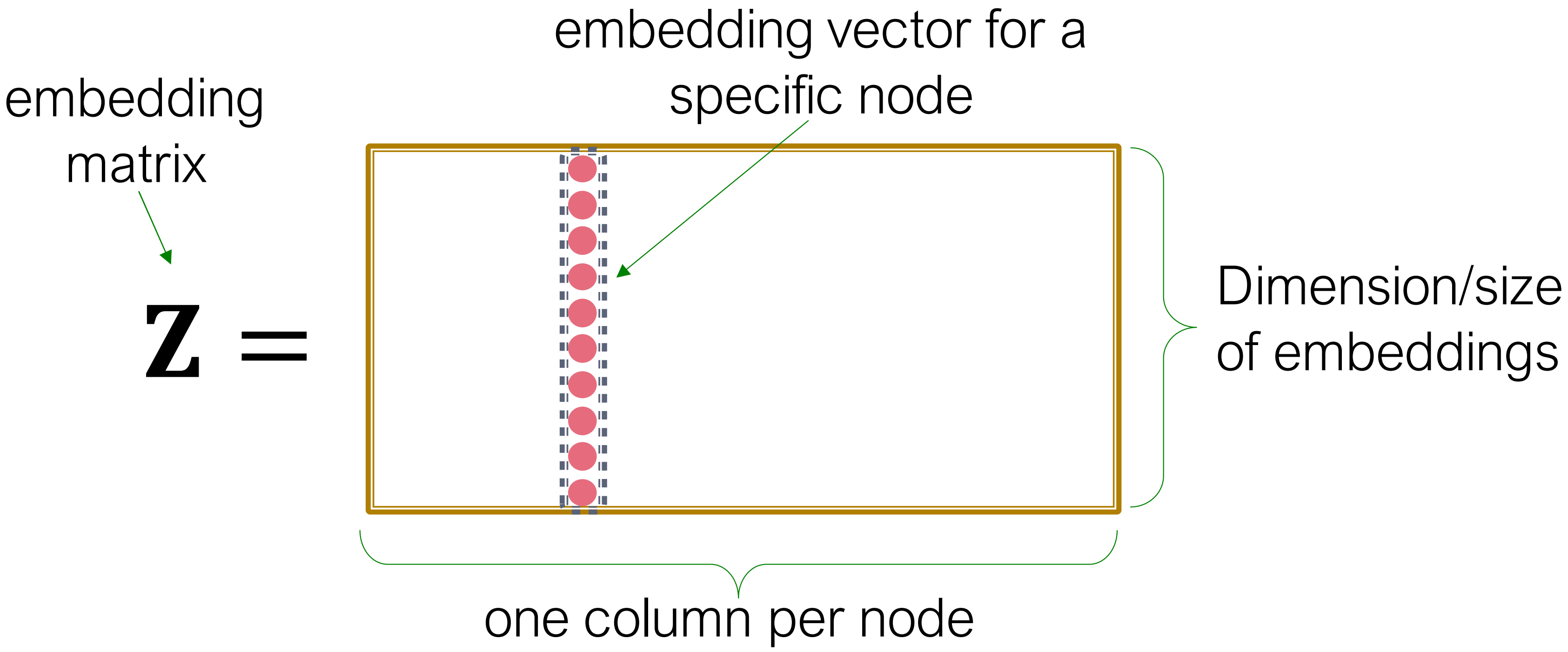
$$\text{ENC}(v) = \mathbf{z}_v = \mathbf{Z} \cdot v$$

$\mathbf{Z} \in \mathbb{R}^{d \times |\mathcal{V}|}$ matrix, each column is a node embedding [what we learn / optimize]

$v \in \mathbb{I}^{|\mathcal{V}|}$ indicator vector, all zeroes except a one in column indicating node *v*

“Shallow” Encoding

Simplest encoding approach: **encoder is just an embedding-lookup**



"Shallow" Encoding

Simplest encoding approach: **Encoder is just an embedding-lookup**

Each node is assigned a unique embedding vector
(i.e., we directly optimize the embedding of each node)

Many methods: DeepWalk, node2vec

Summary so far

- **Encoder + Decoder Framework**
  - Shallow encoder: embedding lookup
  - Parameters to optimize: $\mathbf{Z}$ which contains node embeddings $\mathbf{z}_u$ for all nodes $u \in V$
  - We will cover deep encoders in the GNNs

  - **Decoder:** based on node similarity.
  - **Objective:** maximize $\mathbf{z}_v^{\mathrm{T}} \mathbf{z}_u$ for node pairs (u, v) that are **similar**

Discussion: How to Define Node Similarity?

- Key choice of methods is **how they define node similarity.**

- Should two nodes have a similar embedding if they…
  - are linked?
  - share neighbors?
  - have similar "structural roles"?

Node Embedding: Key Components

- **Encoder:** maps each node to a low-dimensional vector

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

d-dimensional embedding

node in the input graph

- **Similarity function:** specifies how the relationships in vector space map to the relationships in the original network

$$\text{similarity}(u, v) \approx \mathbf{z}_v^{\mathrm{T}} \mathbf{z}_u$$

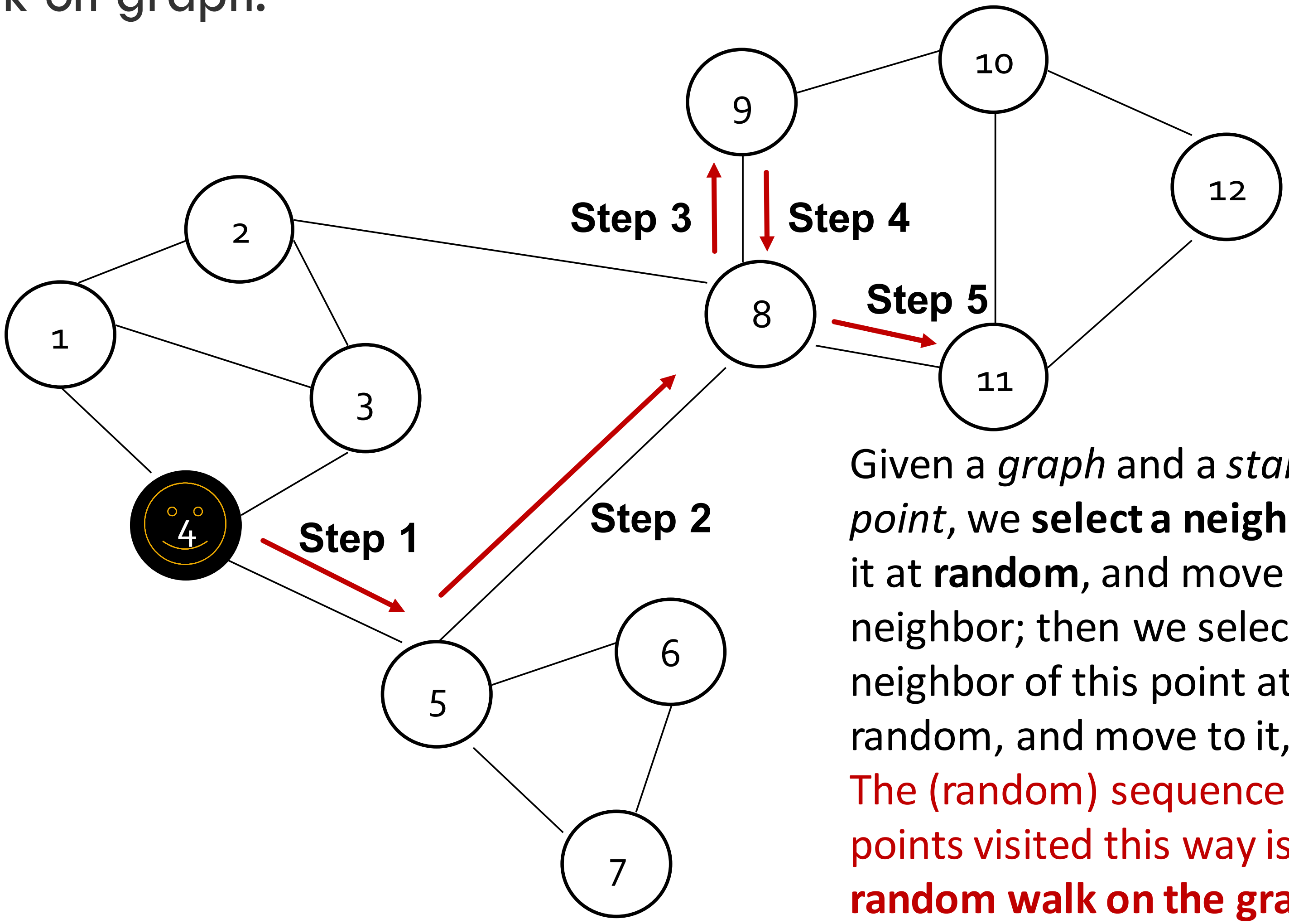
Decoder

Similarity of u and v in the original network

dot product between node embeddings

Similarity Function based on Random Walk

Random walk on graph:



Given a *graph* and a *starting point*, we **select a neighbor** of it at **random**, and move to this neighbor; then we select a neighbor of this point at random, and move to it, etc. The (random) sequence of points visited this way is a **random walk on the graph**.

Similarity Function based on Random Walk

$$\mathbf{z}_u^{\mathrm{T}} \mathbf{z}_v \approx$$ probability that $\mathbf{u}$ and $\mathbf{v}$ co-occur on a random walk over the graph

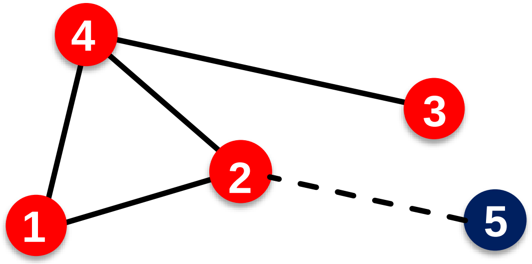
Why Random Walk?

1. **Expressivity:** Flexible stochastic definition of node similarity that incorporates both local and higher-order neighborhood information
   Idea: if random walk starting from node u visits v with high probability, u and v are similar (high-order multi-hop information)

2. **Efficiency:** Do not need to consider all node pairs when training; only need to consider pairs that co-occur on random walks

Limitations of Random Walk Embedding (1)

- Cannot obtain embeddings for nodes not in the training set



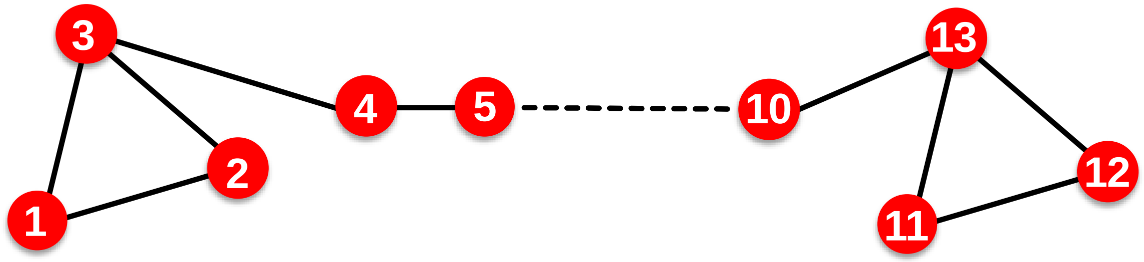
Training set

A newly added node 5 at test time (e.g., new user in a social network)

Cannot compute its embedding with DeepWalk / node2vec. Need to recompute all node embeddings.

Limitations of Random Walk Embedding (2)

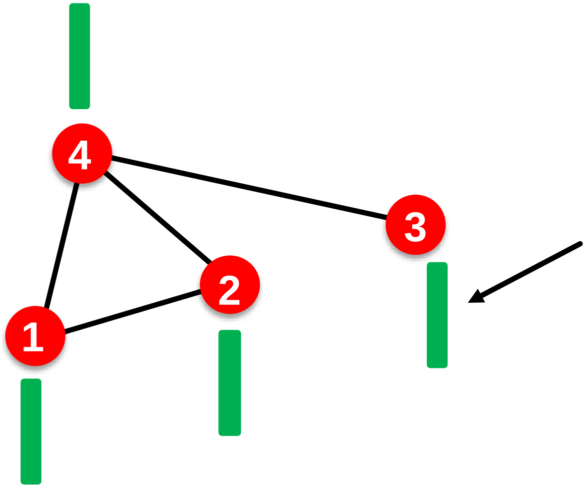
- Cannot capture **structural similarity**:



- Node 1 and 11 are **structurally similar** – part of one triangle, degree 2, …
- However, they have very **different** embeddings.
  - It's unlikely that a random walk will reach node 11 from node 1.

Limitations of Random Walk Embedding (3)

- Cannot utilize node, edge and graph features



Feature vector
(e.g. protein properties in a protein-protein interaction graph)

DeepWalk / node2vec embeddings do not incorporate such node features

Solution to these limitations: Deep Representation Learning and Graph Neural Networks

Summary

- **Encoder + Decoder Framework**
  - Shallow encoder: embedding lookup
  - Parameters to optimize: $\mathbf{Z}$ which contains node embeddings $\mathbf{z}_u$ for all nodes $u \in V$
  - We will cover deep encoders in the GNNs

  - **Decoder:** based on node similarity.
  - **Objective:** maximize $\mathbf{z}_v^{\mathrm{T}} \mathbf{z}_u$ for node pairs (u, v) that are **similar**

Discussion: How to Define Node Similarity?

- Key choice of methods is **how they define node similarity.**

- Should two nodes have a similar embedding if they…
  - are linked?
  - share neighbors?
  - have similar “structural roles”?

Graph Neural Networks (GNNs)

Slides adapted from:

- Jure Leskovec, Stanford CS224W: Machine Learning with Graphs

Deep Graph Encoders

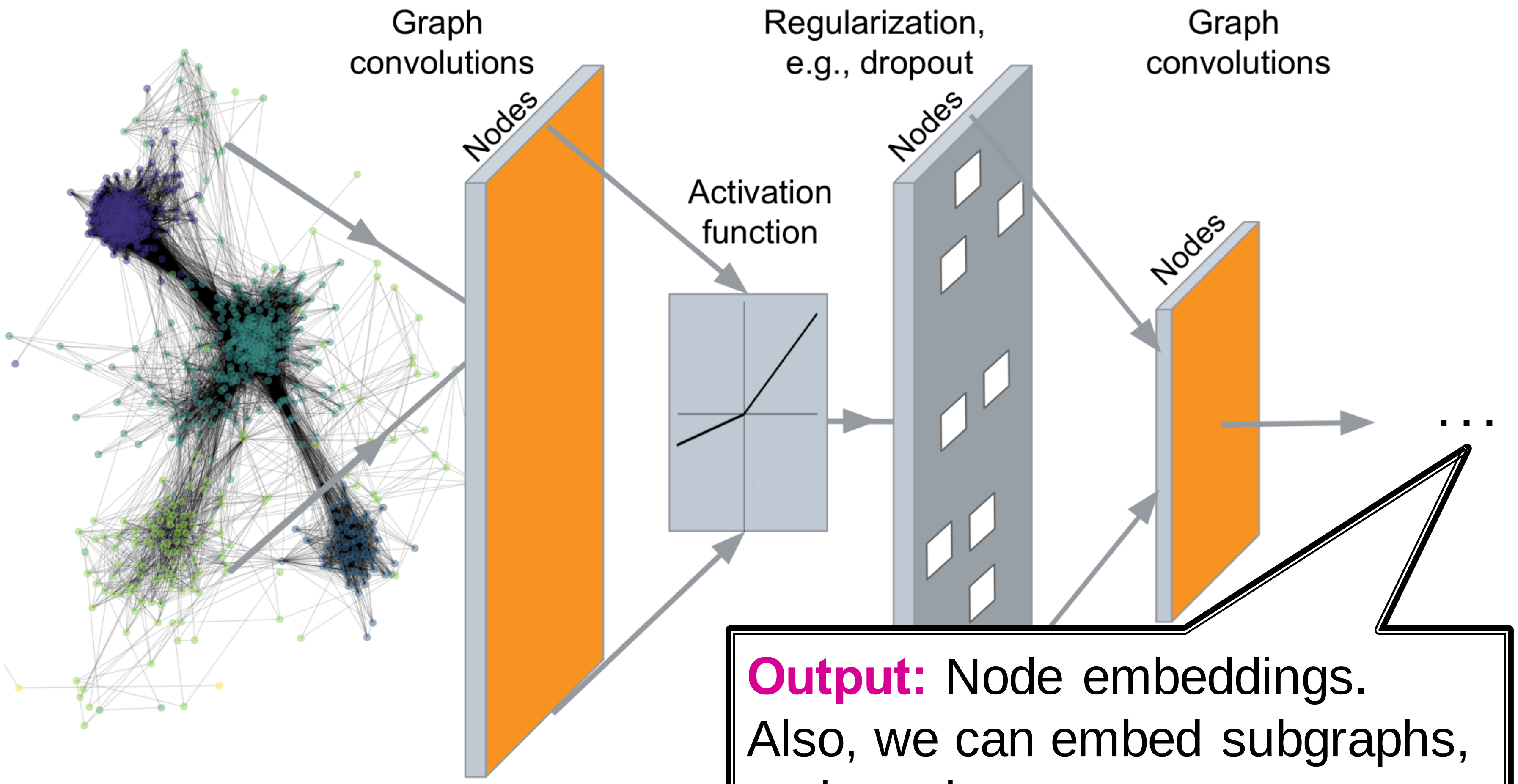
- Encoding based on graph neural networks

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

v.s. **Shallow Encoder:**

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

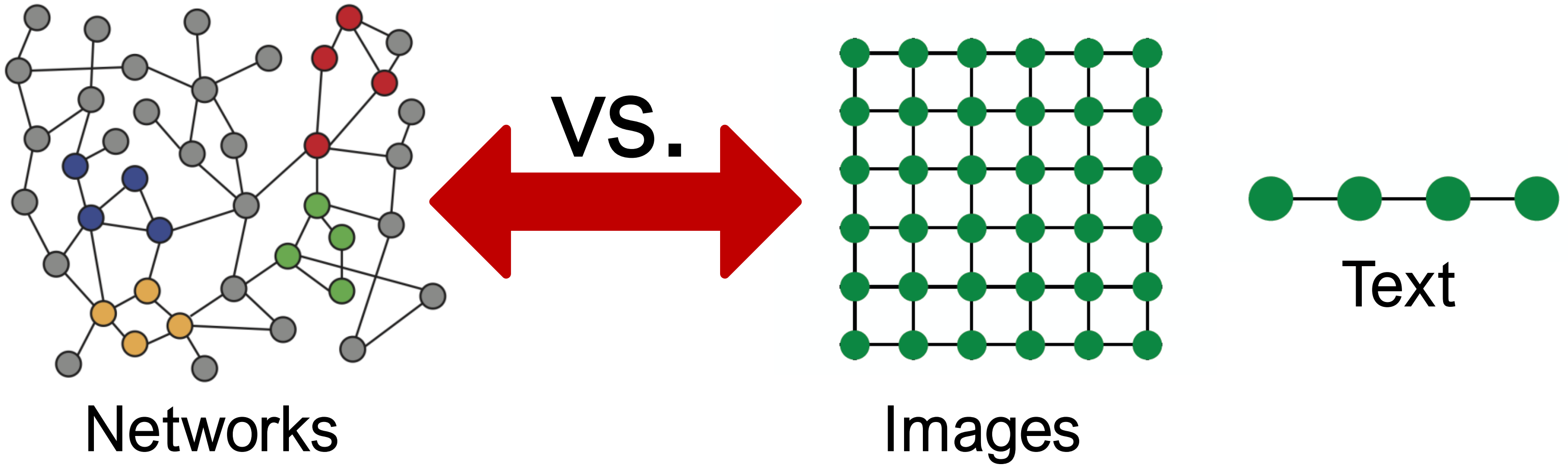
Deep Graph Encoders



Output: Node embeddings. Also, we can embed subgraphs, and graphs

Graphs are more complex than images / text

- Arbitrary size and complex topological structure (i.e., no spatial locality like grids)



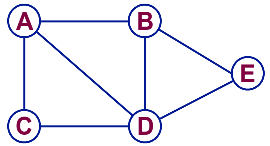
- No fixed node ordering or reference point
- Often dynamic and have multimodal features

Graph Neural Networks: Setup

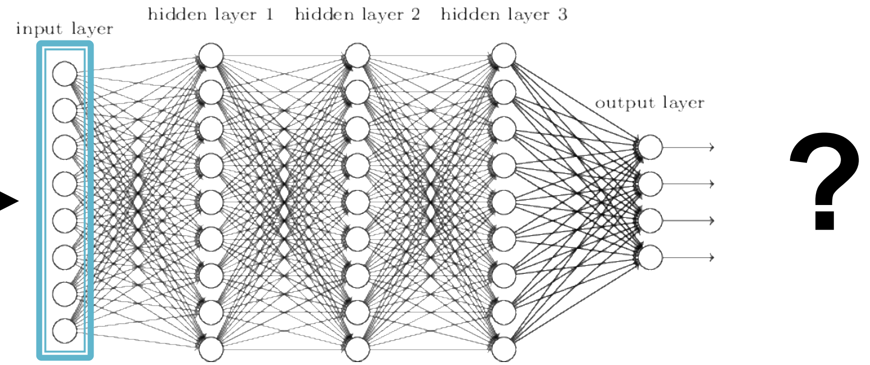
- **Assume we have a graph G:**
  - V is the **vertex set**
  - $\boldsymbol{A}$ is the **adjacency matrix** (assume binary)
  - $\boldsymbol{X} \in \mathbb{R}^{|V| \times d}$ is a matrix of **node features**
  - v: a node in V; $N(v)$: the set of neighbors of v.
  - **Node features:**
    - Social networks: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



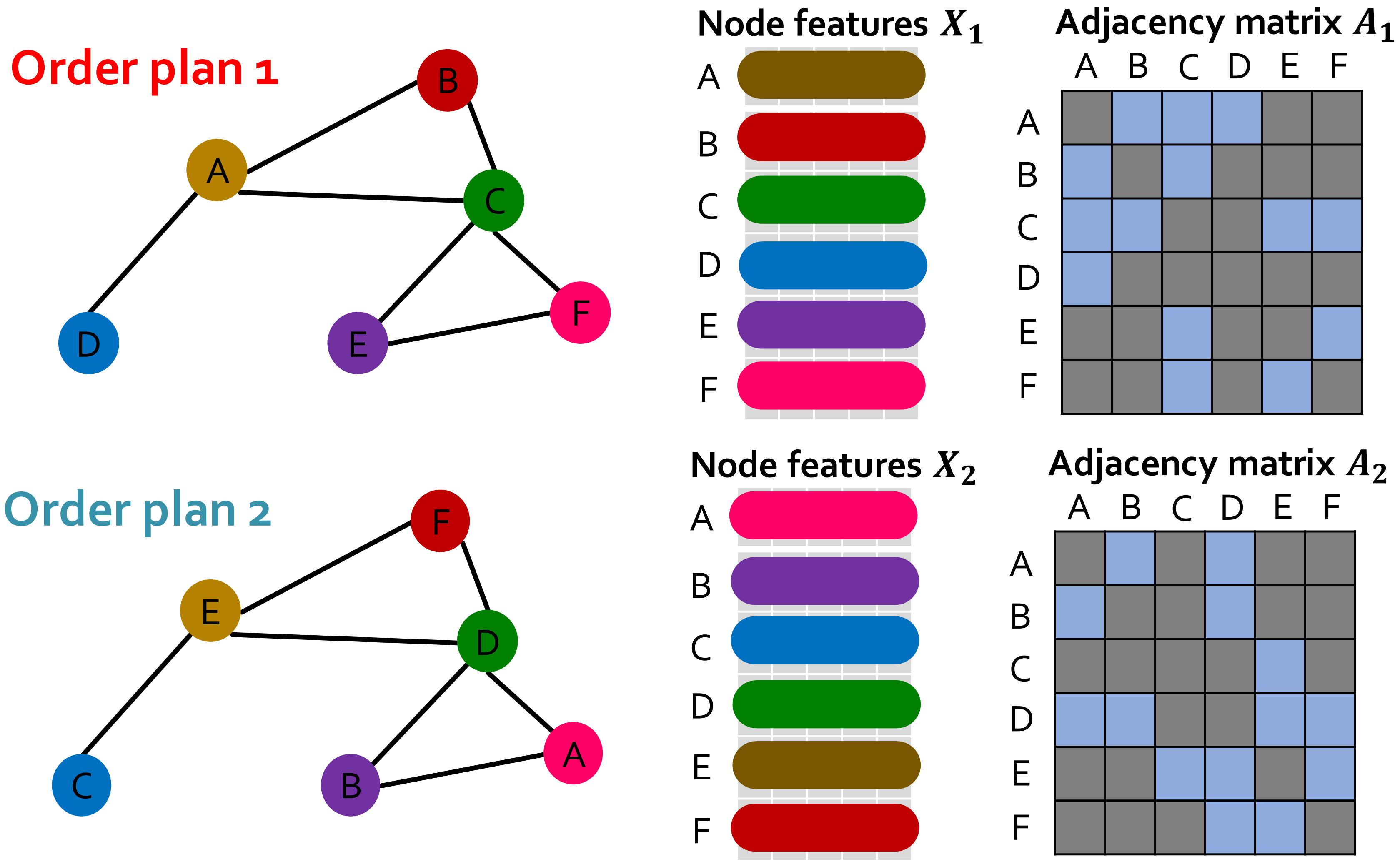
	A	B	C	D	E	Feat	
A	0	1	1	1	0	1	0
B	1	0	0	1	1	0	0
C	1	0	0	1	0	0	1
D	1	1	1	0	1	1	1
E	0	1	0	1	0	1	0



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

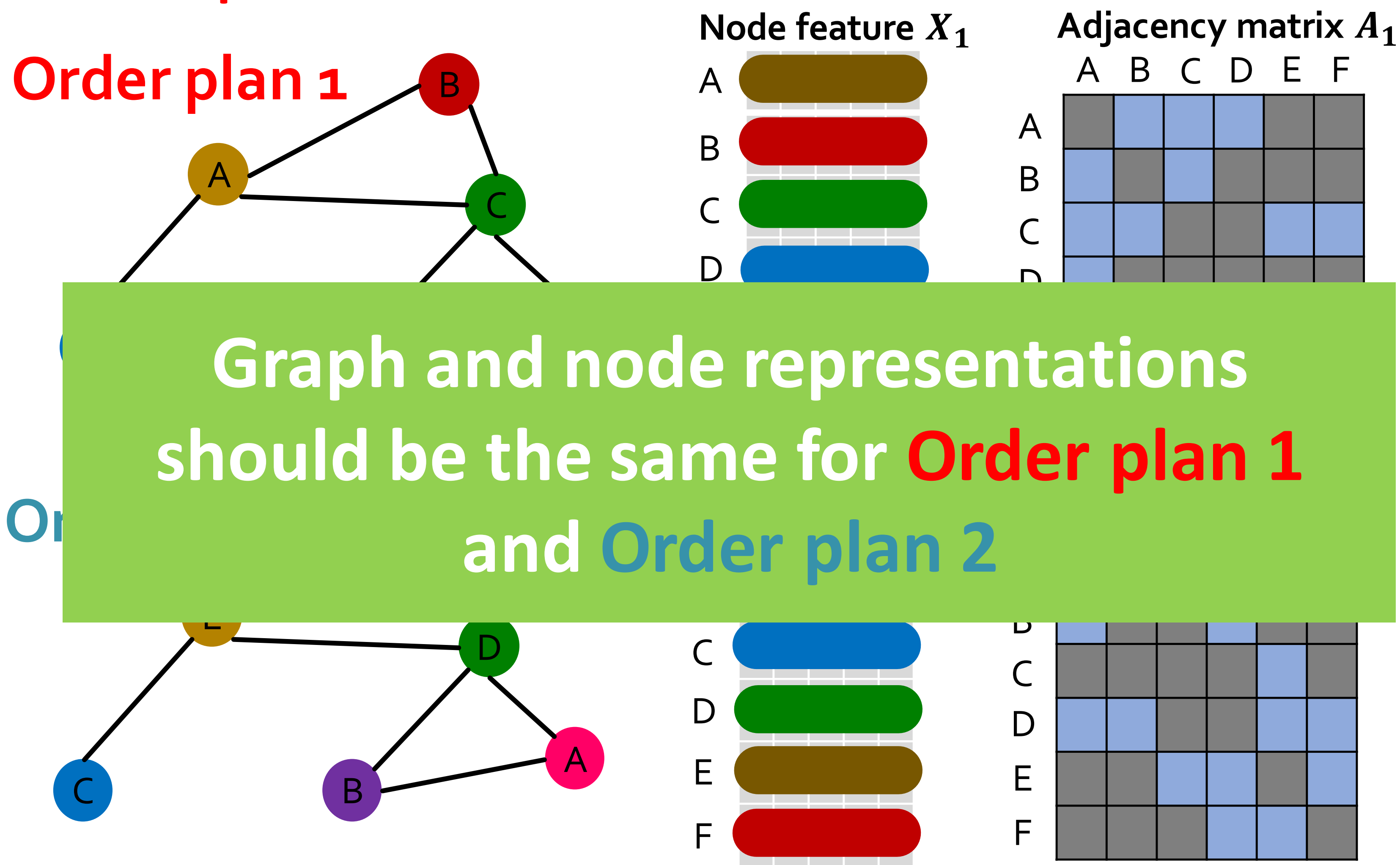
Permutation Invariance

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



Permutation Invariance

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



Graph and node representations should be the same for Order plan 1 and Order plan 2

Permutation Invariance

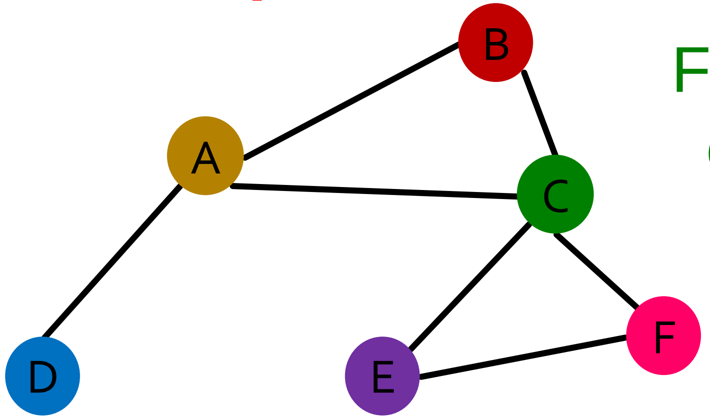
What does it mean by "graph representation is same for two order plans"?

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

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

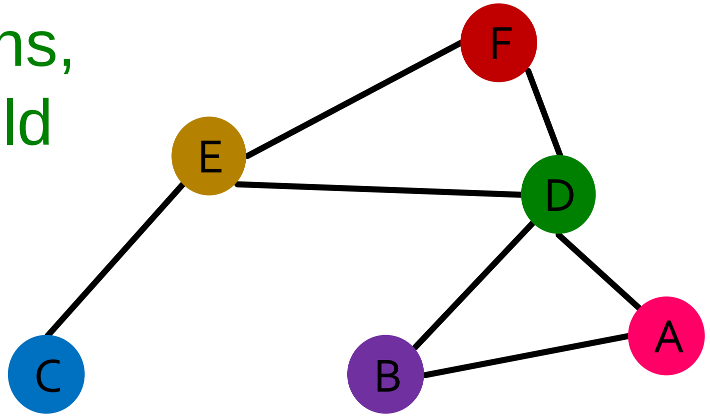
$\boldsymbol{A}$ is the adjacency matrix
$\boldsymbol{X}$ is the node feature matrix

Order plan 1: A_1, X_1



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

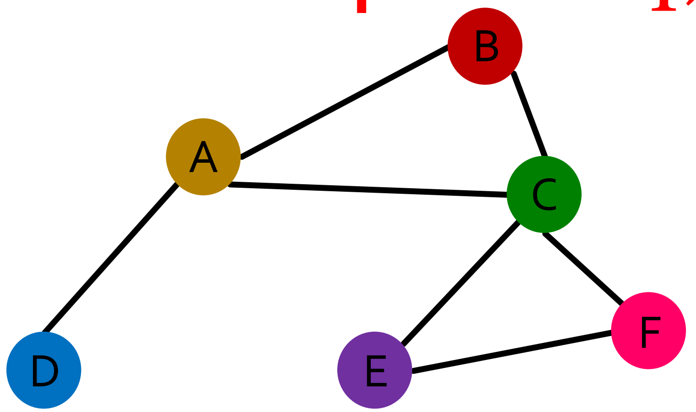
Order plan 2: A_2, X_2



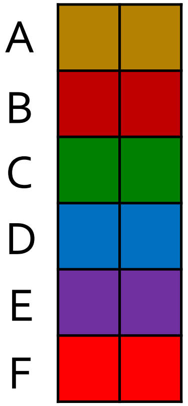
Permutation Equivariance

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

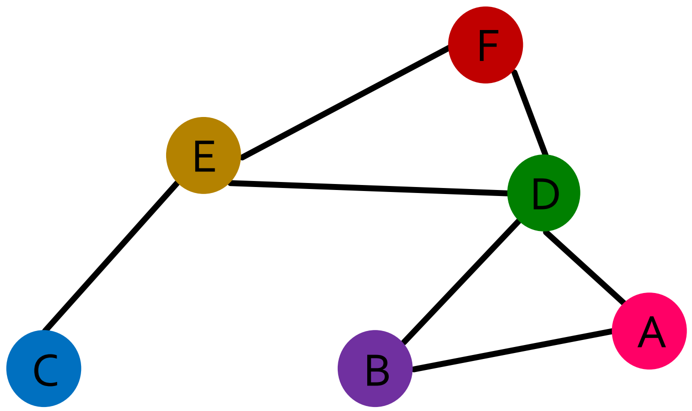
Order plan 1: $\boldsymbol{A}_1, \boldsymbol{X}_1$



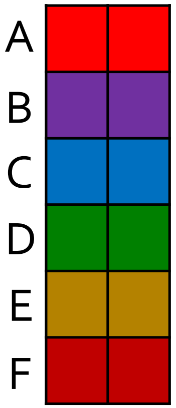
$f(\boldsymbol{A}_1, \boldsymbol{X}_1) =$



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



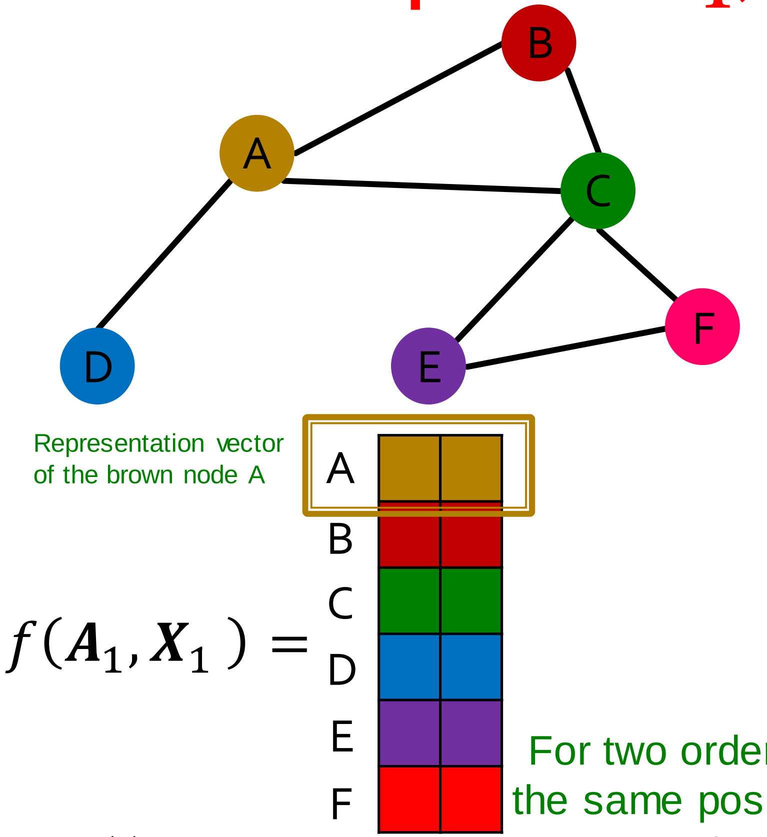
$f(\boldsymbol{A}_2, \boldsymbol{X}_2) =$



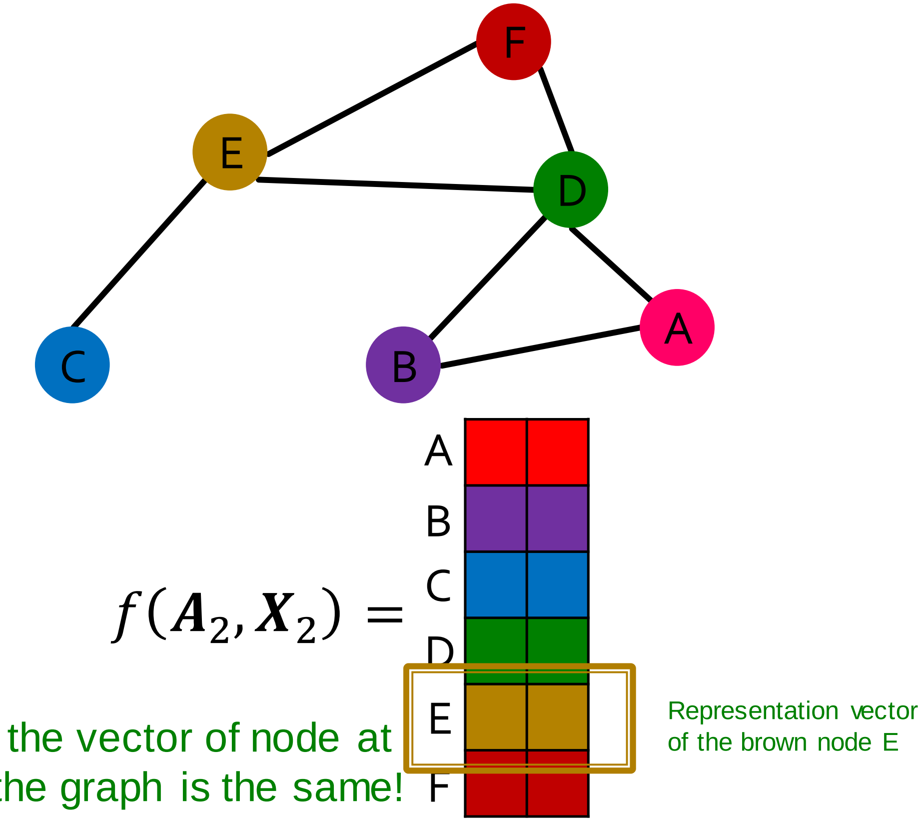
Permutation Equivariance

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



Order plan 2: A_2, X_2

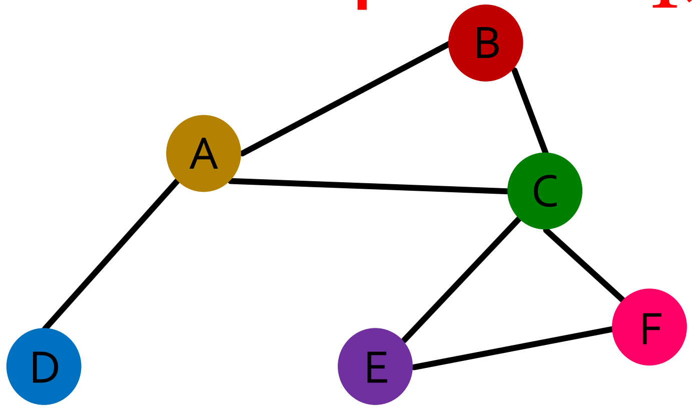


For two order plans, the vector of node at the same position in the graph is the same!

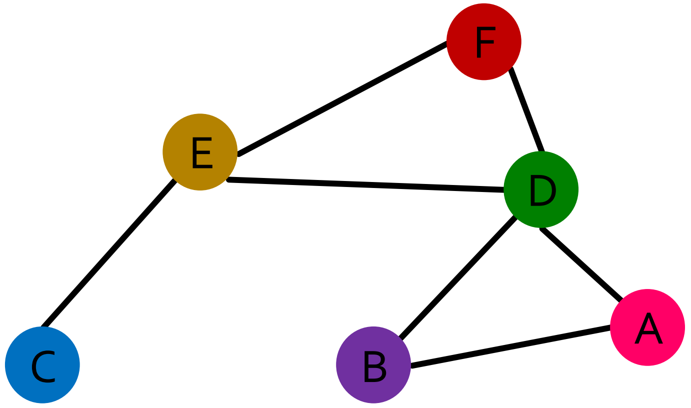
Permutation Equivariance

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

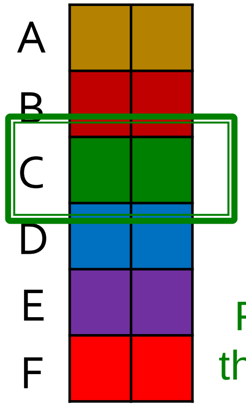
Order plan 1: $\boldsymbol{A}_1, \boldsymbol{X}_1$



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

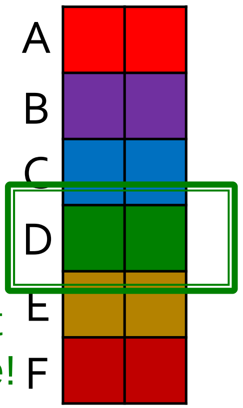


$f(\boldsymbol{A}_1, \boldsymbol{X}_1) =$



Representation vector of the green node C

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



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!

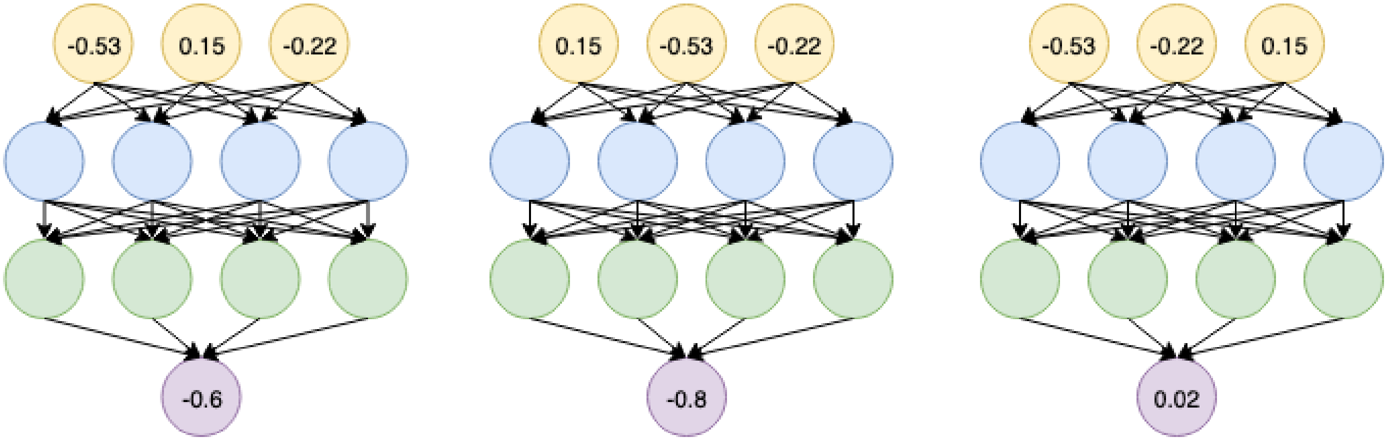
Graph Neural Networks Overview

- 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!

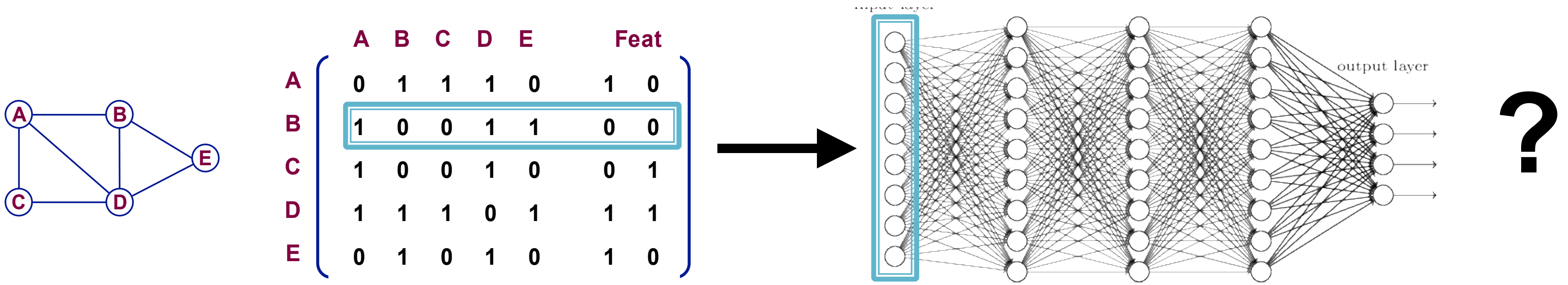


Graph Neural Networks Overview

- GNNs consist of multiple permutation equivariant / invariant functions

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

- **No.**



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

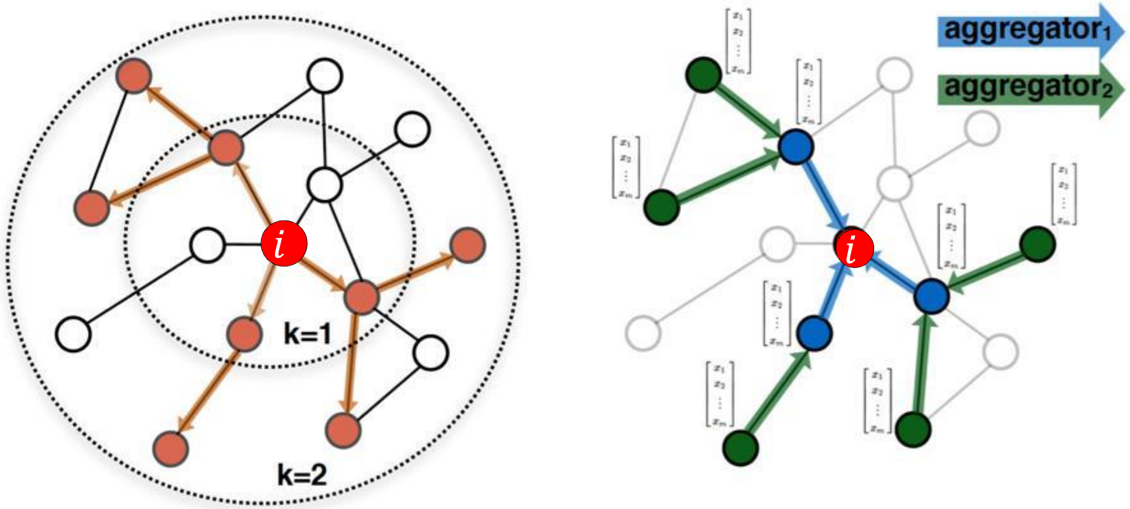
Graph Neural Networks Overview

- GNNs consist of multiple permutation equivariant / invariant functions

- Next: 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

Questions?