COMP4222 Machine Learning with Structured Data

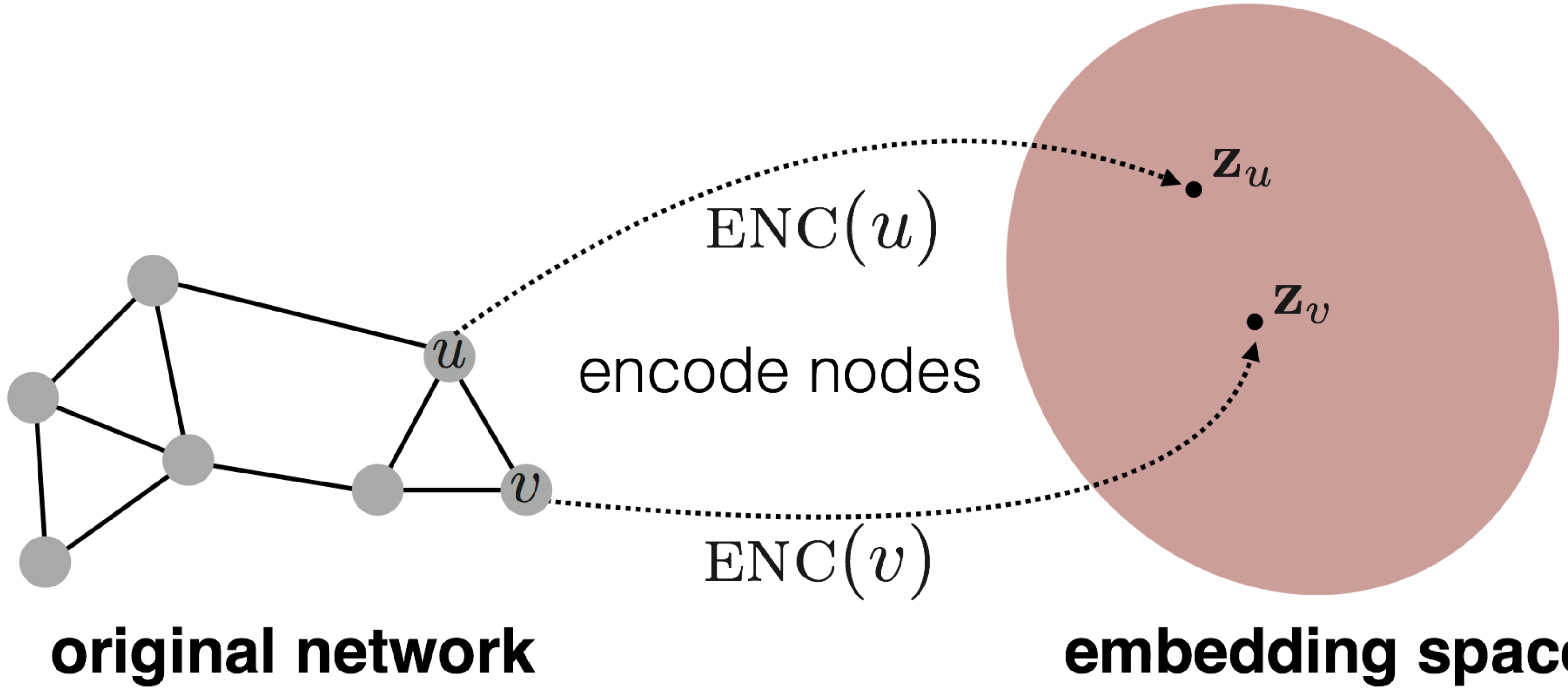
Graph Neural Networks 1

Instructor: Yangqiu Song

Slides credits: Jure Leskovec, William L. Hamilton, Rex Ying, Rok Sosic

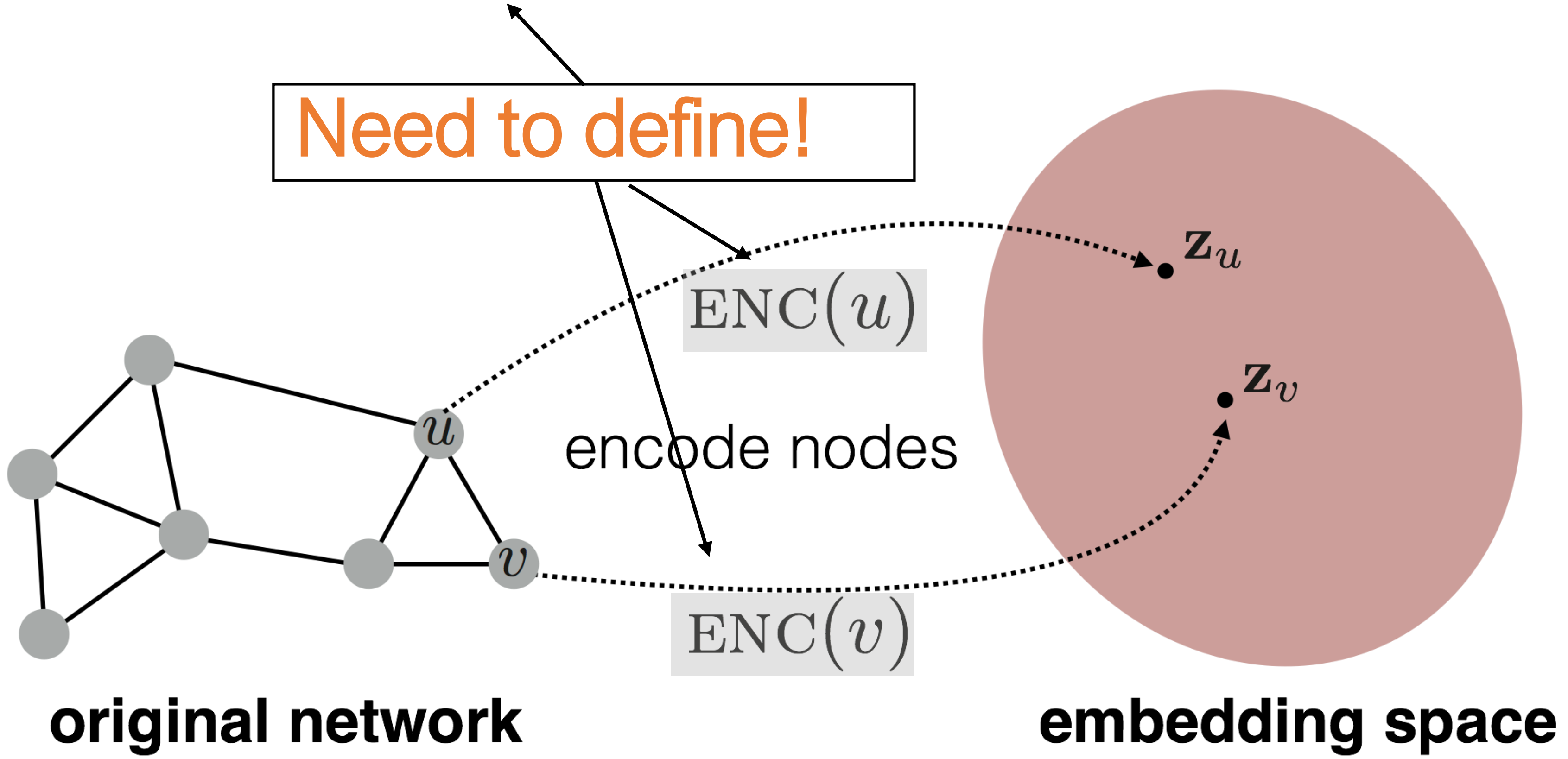
Embedding Nodes

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



Embedding Nodes

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



Two 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 relationships in vector space map to relationships in the original network.

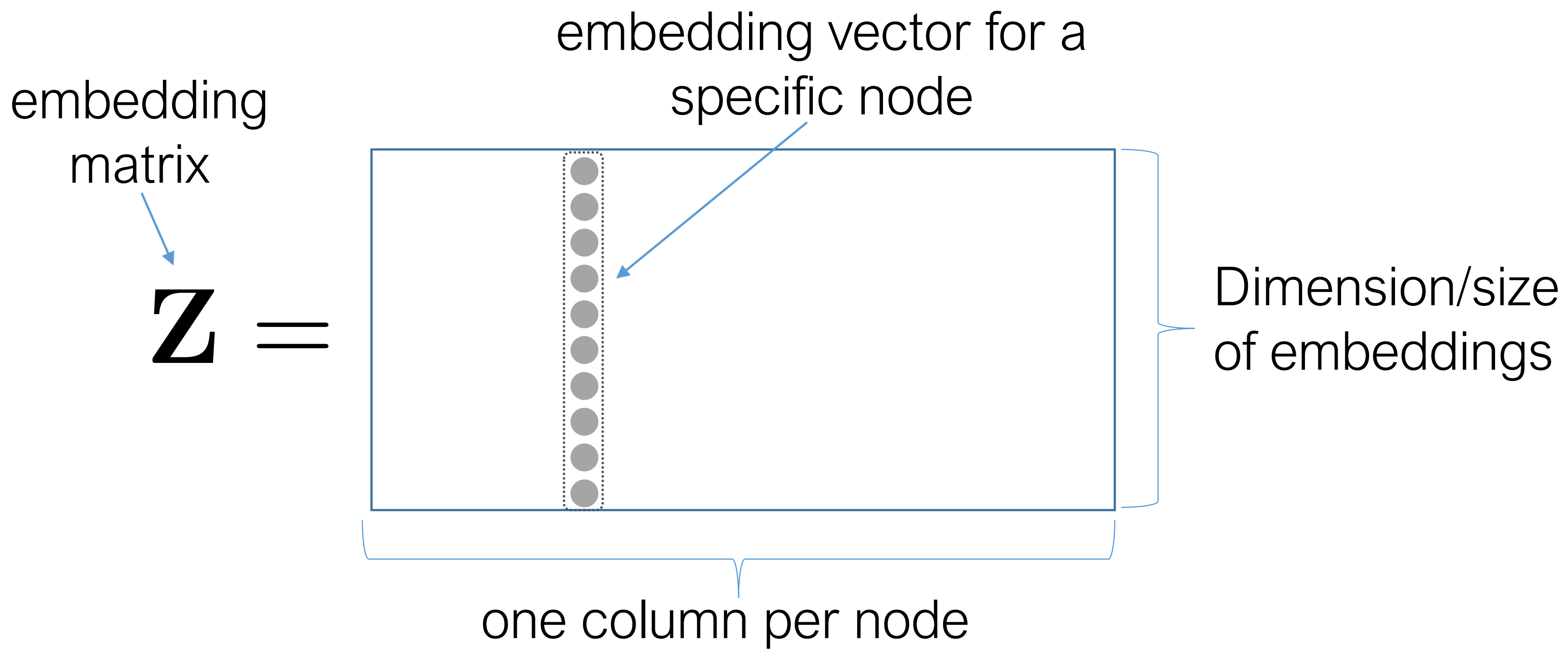
$$\text{similarity}(u, v) \approx \mathbf{z}_v^\top \mathbf{z}_u$$

Similarity of u and v in the original network

dot product between node embeddings

Shallow Encoders

- So far we have focused on **"shallow" encoders**, i.e. embedding lookups:



embedding matrix
$\mathbf{Z} =$
embedding vector for a specific node
Dimension/size of embeddings
one column per node

Shallow Encoders

- **Limitations** of shallow embedding methods:
  - **$O(|V|)$ parameters are needed**:
    - No sharing of parameters between nodes
    - Every node has its own unique embedding
  - **Inherently "transductive"**:
    - Cannot generate embeddings for nodes that are not seen during training
  - **Do not incorporate node features**:
    - Many graphs have features that we can and should leverage

From “Shallow” to “Deep”

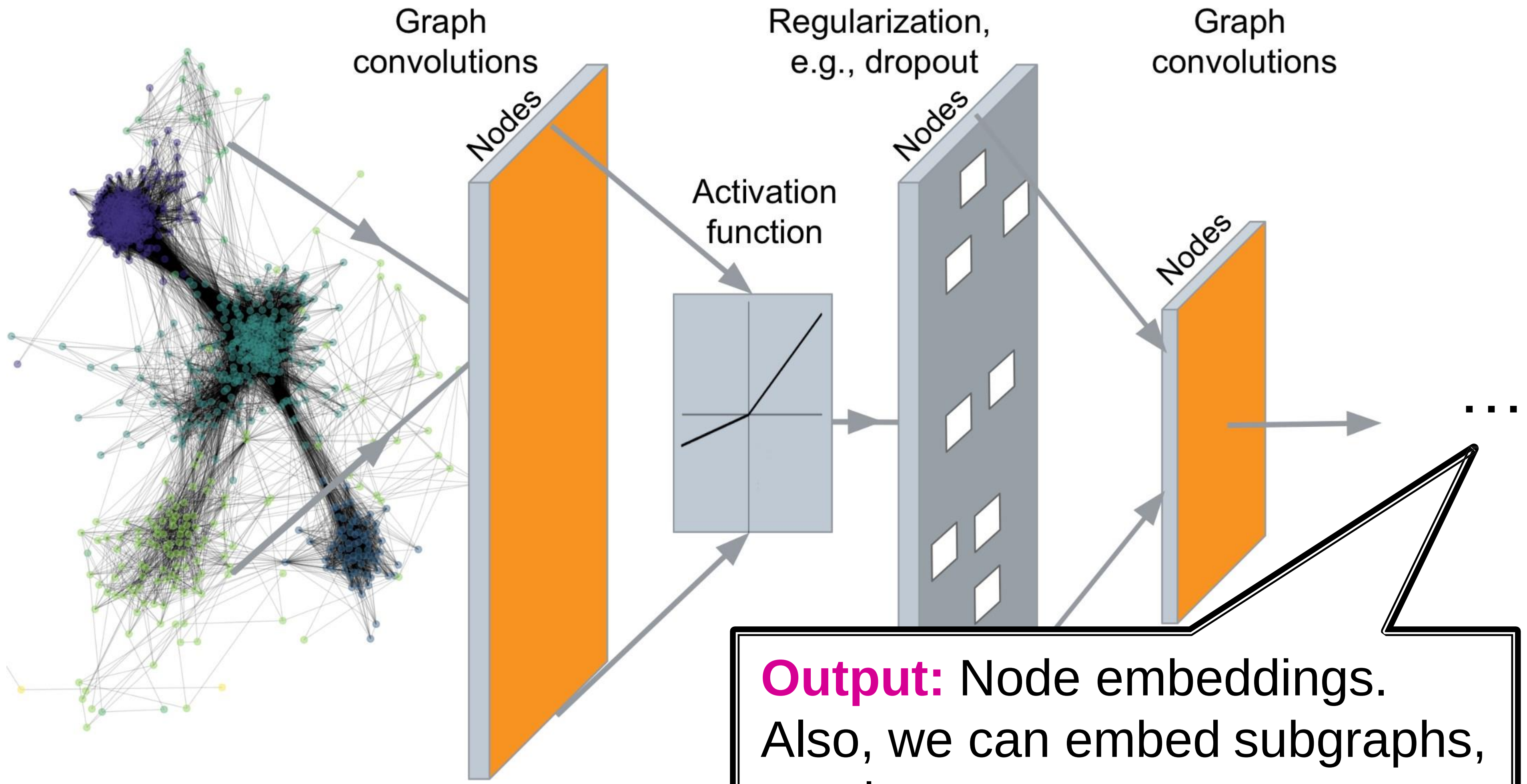
- We will now discuss “deeper” methods based on **graph neural networks.**

$$\mathrm{ENC}(v) =$$

- complex function that depends on graph structure
- multiple layers of non-linear transformations based on graph structure

- In general, all of these more complex encoders can be combined with the similarity functions from the previous section.

Deep Graph Encoders



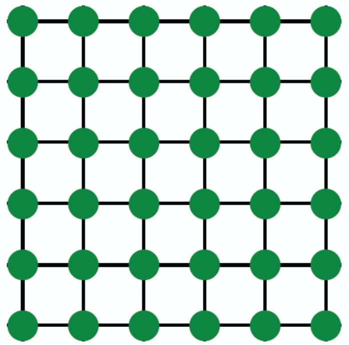
Output: Node embeddings. Also, we can embed subgraphs, graphs

Tasks on Networks

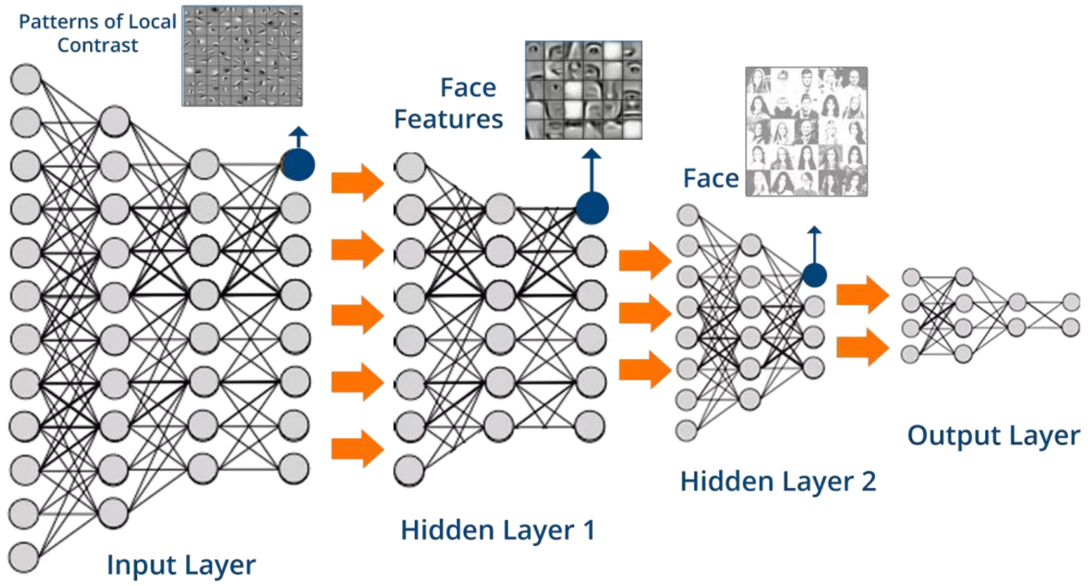
- Node classification
  - Predict a type of a given node
- Link prediction
  - Predict whether two nodes are linked
- Community detection
  - Identify densely linked clusters of nodes
- Network similarity
  - How similar are two (sub)networks

Modern Machine Learning

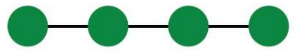
- Modern deep learning toolbox is designed for simple sequences & grids

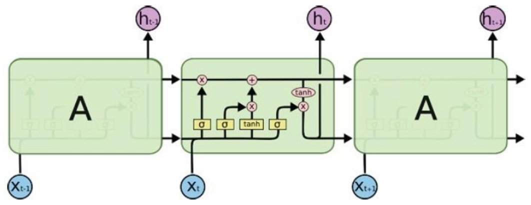


Images



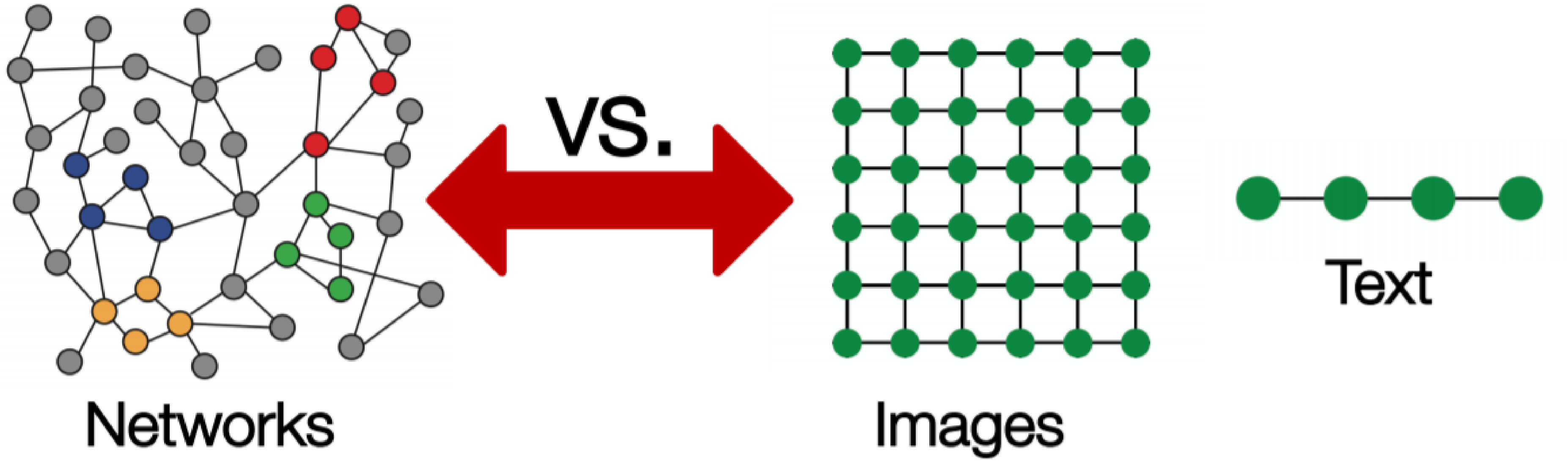
Text/Speech





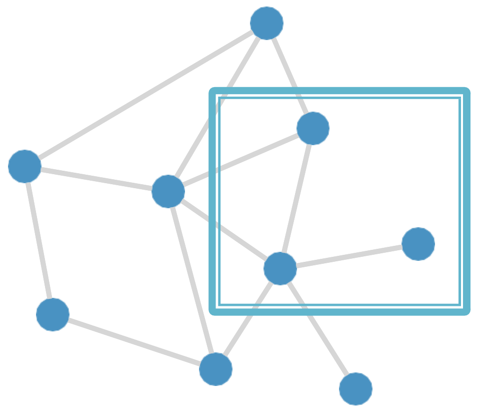
Networks are Far More Complex!

- 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

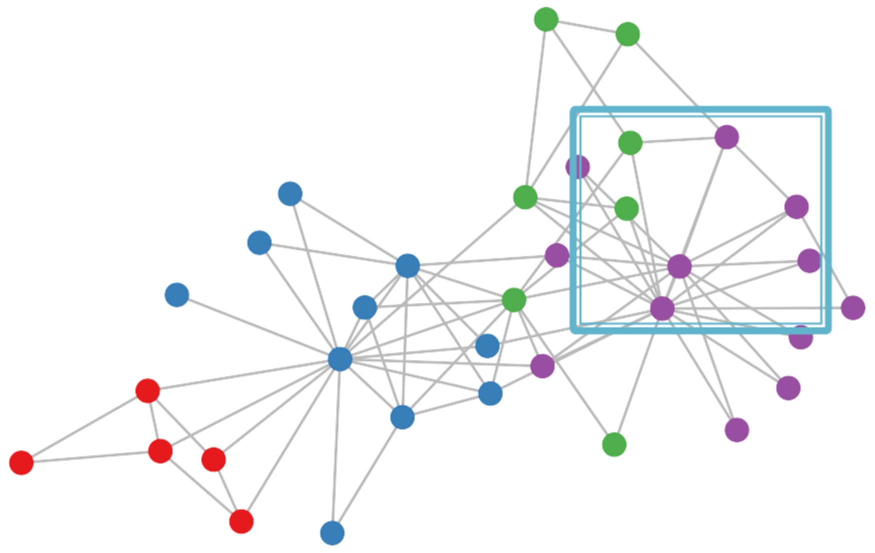


Real-World Graphs

- **But our graphs look like this:**



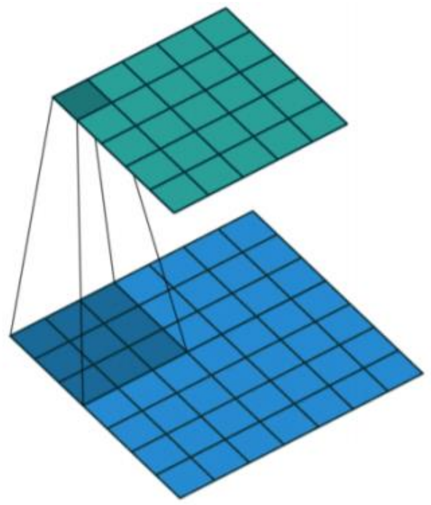
or this:



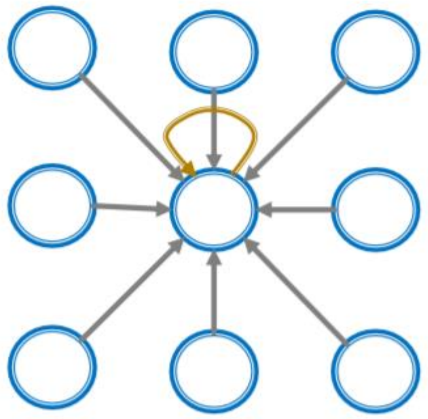
- There is no fixed notion of locality or sliding window on the graph
- Graph is permutation invariant

From Images to Graphs

- Single CNN layer with 3x3 filter



Image

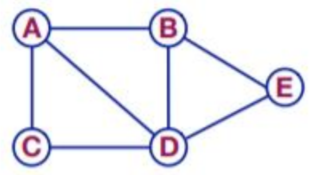


Graph

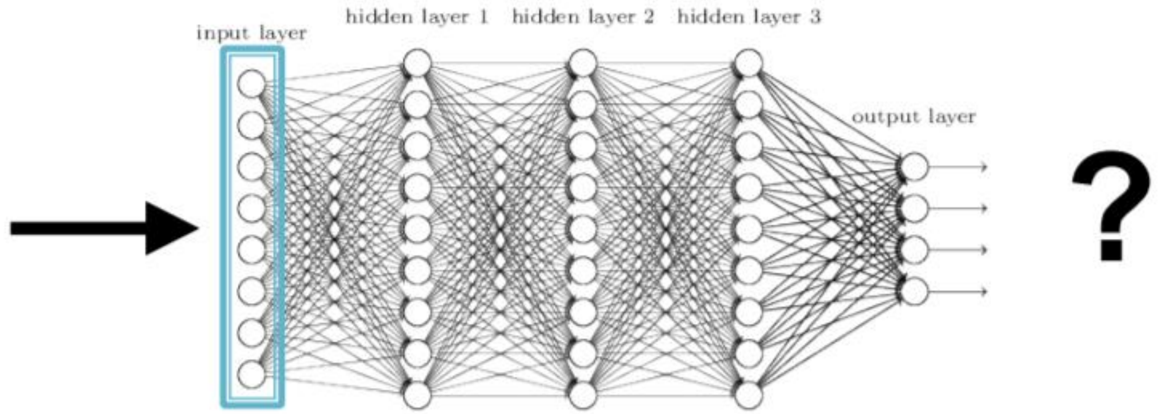
- Transform information at the neighbors and combine it:
  - Transform "messages" h_i from neighbors: $W_i h_i$
  - Add them up: $\sum_i W_i h_i$

A Naïve Approach

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



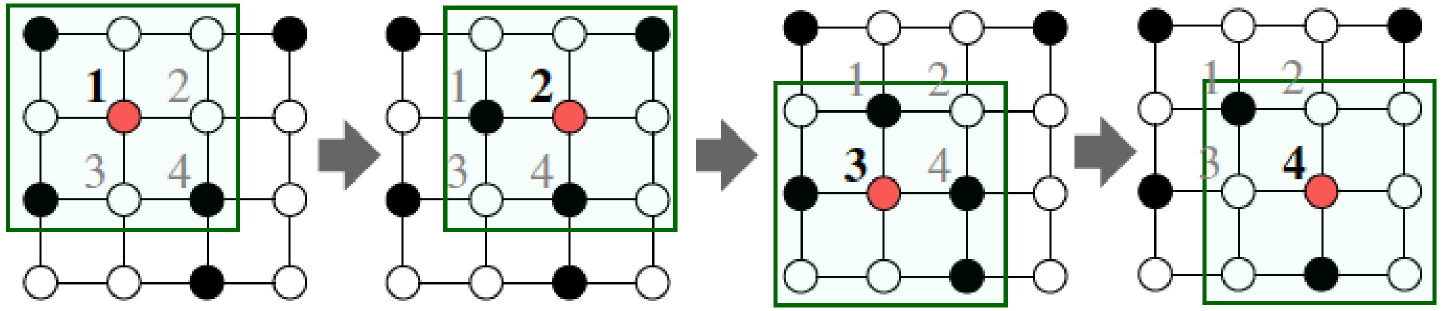
$$\begin{array}{c|ccccc|cc} & A & B & C & D & E & \multicolumn{2}{c}{\text{Feat}} \\ A & 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ B & \boxed{1} & 0 & 0 & 1 & 1 & 0 & 0 \\ C & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ D & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ E & 0 & 1 & 0 & 1 & 0 & 1 & 0 \end{array}$$

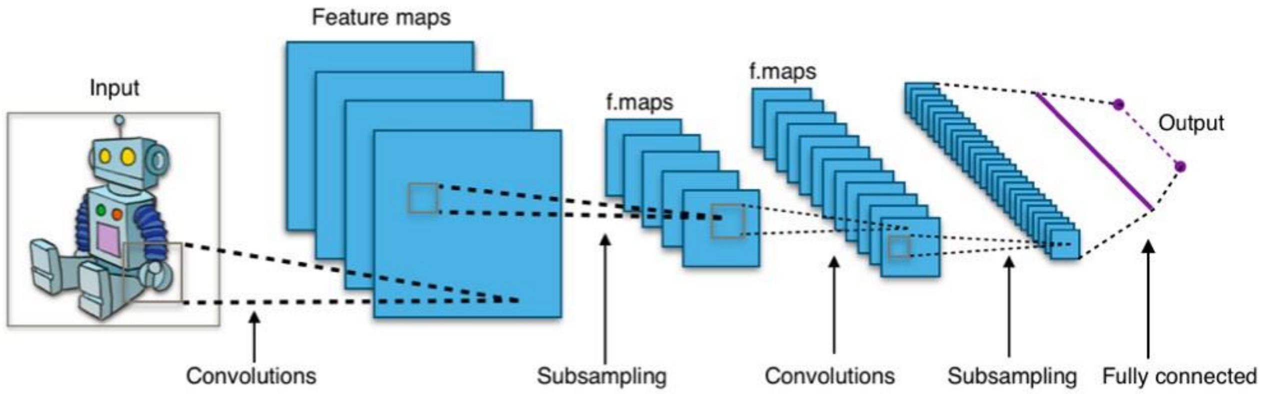


- Issues with this idea:
  - O(N) parameters
  - Not applicable to graphs of different sizes
  - Not invariant to node ordering

General Idea: Graph Convolution

- **CNN on an image:**





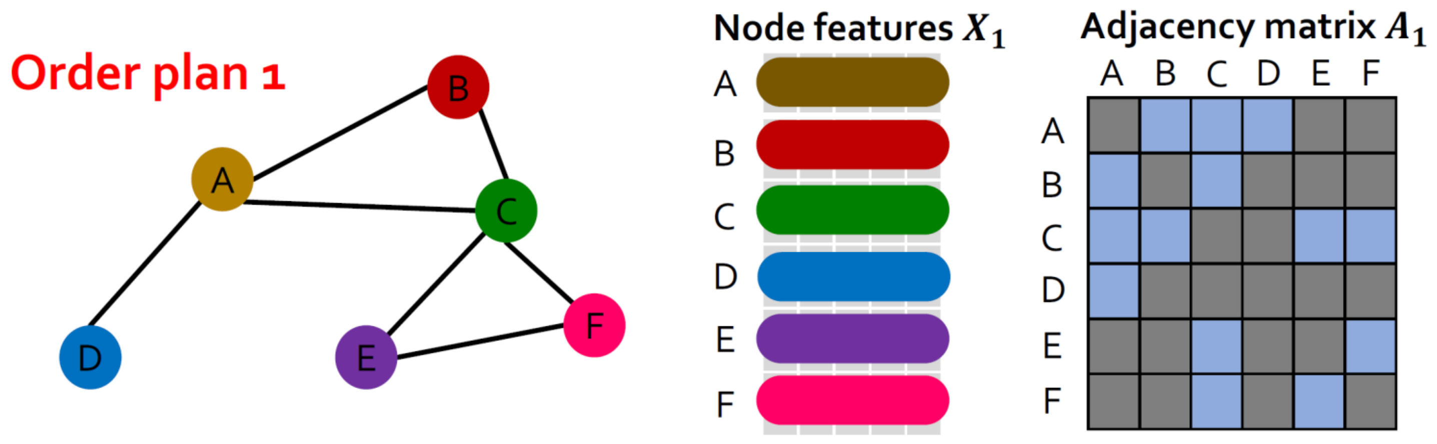
Goal is to generalize convolutions beyond simple lattices
Leverage node features/attributes (e.g., text, images)

Setup

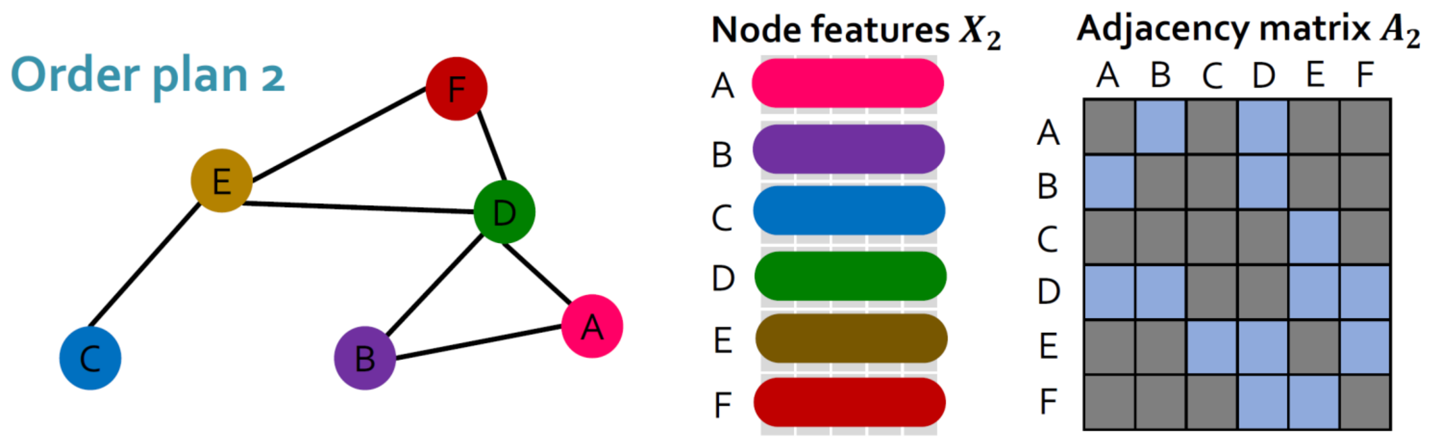
- Assume we have a graph G:
  - V is the vertex set.
  - $\boldsymbol{A}$ is the adjacency matrix (assume binary).
  - v is a node in V; $N(v)$ is the set of neighbors of v
  - **$\boldsymbol{X} \in \mathbf{R}^{m \times |V|}$ is a matrix of node features.**
    - Attributes, text, image data, E.g.,
      - Social network: user profile, user image
      - Biological network: gene expression profiles, functional information
    - Node degrees, clustering coefficients, etc.
    - Indicator vectors (i.e., one-hot encoding of each node)
      - Retrieve from an embedding matrix which is learnable

Permutation Invariance

- **Graph does not have a canonical order of the nodes!**
- We can have many different order plans.



	A	B	C	D	E	F
A	⬛	🟦	🟦	🟦	⬛	⬛
B	🟦	⬛	🟦	⬛	⬛	⬛
C	🟦	🟦	⬛	⬛	🟦	🟦
D	🟦	⬛	⬛	⬛	⬛	⬛
E	⬛	⬛	🟦	⬛	⬛	🟦
F	⬛	⬛	🟦	⬛	🟦	⬛



	A	B	C	D	E	F
A	⬛	🟦	⬛	🟦	⬛	⬛
B	🟦	⬛	⬛	🟦	⬛	⬛
C	⬛	⬛	⬛	⬛	🟦	⬛
D	🟦	🟦	⬛	⬛	🟦	🟦
E	⬛	⬛	🟦	🟦	⬛	🟦
F	⬛	⬛	⬛	🟦	🟦	⬛

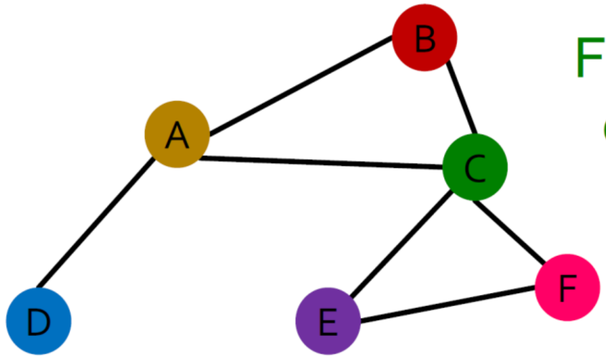
Graph and node representations should be the same for Order plan 1 and Order plan 2

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

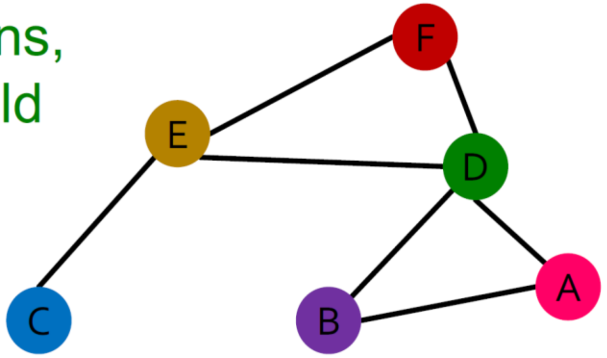
$$f(\boldsymbol{A}_1, \boldsymbol{X}_1) = f(\boldsymbol{A}_2, \boldsymbol{X}_2)$$

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 $\mathbb{R}^d$ then

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

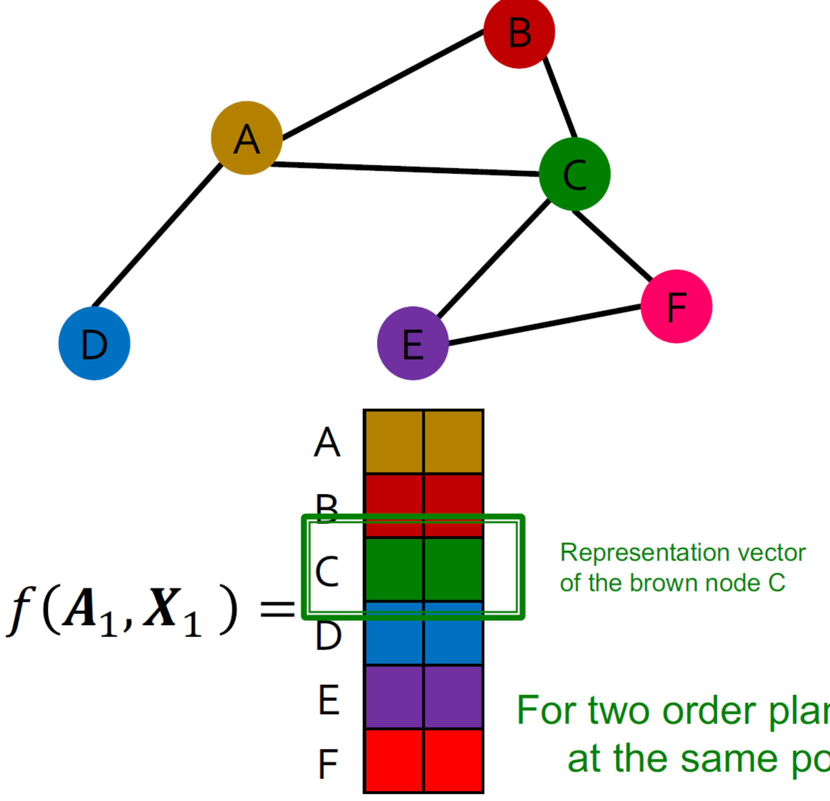
- Then if $f(\boldsymbol{A}_i, \boldsymbol{X}_i) = f(\boldsymbol{A}_j, \boldsymbol{X}_j)$ for any order plan i and j, we formally say $\boldsymbol{f}$ is a **permutation invariant function**.
    - For a graph with m nodes, there are $m!$ different order plans.

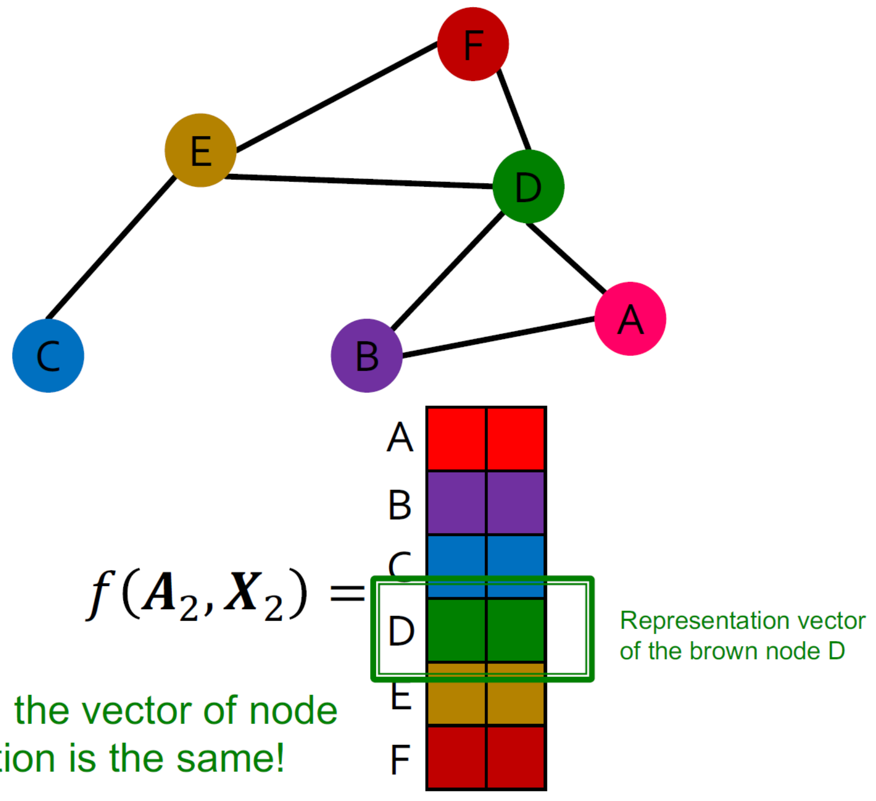
Permutation Equivariance

- **Similarly for node representation:** We learn a function f that maps nodes of G to a matrix $\mathbb{R}^{m*d}$

Order plan 1: A_1, X_1

Order plan 2: A_2, X_2





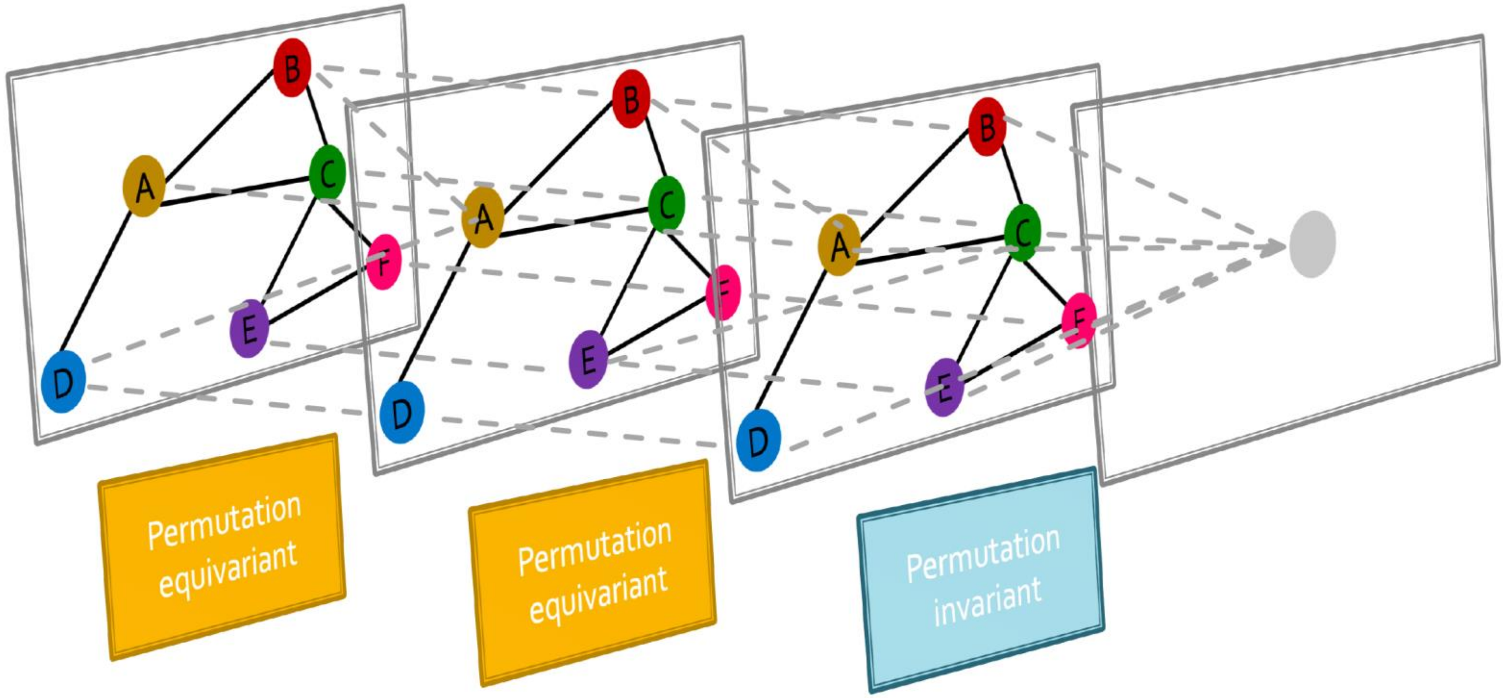
For two order plans, the vector of node at the same position is the same!

Permutation Equivariance

- **For node representation**
  - Consider we learn a function f that maps a graph G = $(\boldsymbol{A}, \boldsymbol{X})$ to a matrix $\mathbb{R}^{m*d}$
    - graph has m nodes, each row is the embedding of a node.
  - For two order plans, the vector of node at the same position is the same!
- Similarly, if this property holds for any pair of order plan i and j, we say f is a **permutation equivariant function**.

Graph Neural Network Overview

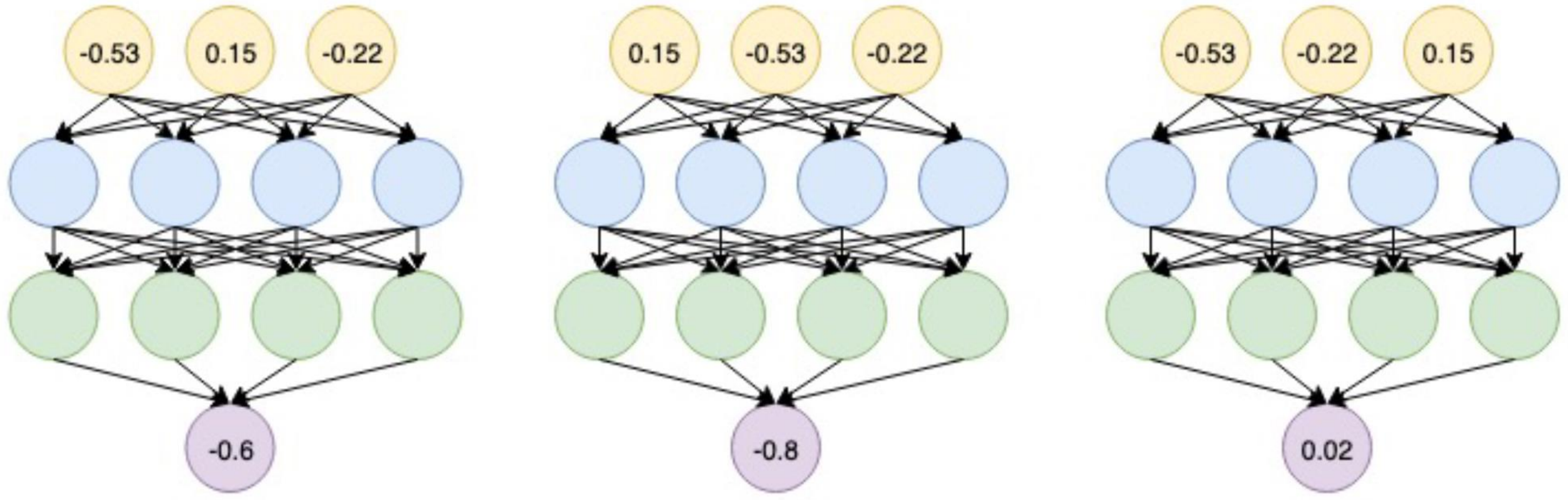
- **Graph neural networks consist of multiple permutation equivariant / invariant functions**



[Bronstein, ICLR 2021 keynote]

Graph Neural Network Overview

- **Are other neural network architectures, e.g., MLPs, permutation invariant / equivariant?**
  - **No.**



Switching the order of the input leads to different outputs!

Outline for this Section

- We will now discuss "deeper" methods based on **graph neural networks.**
    1. **The Basics**
    2. **GraphSAGE**

The Basics: Graph Neural Networks

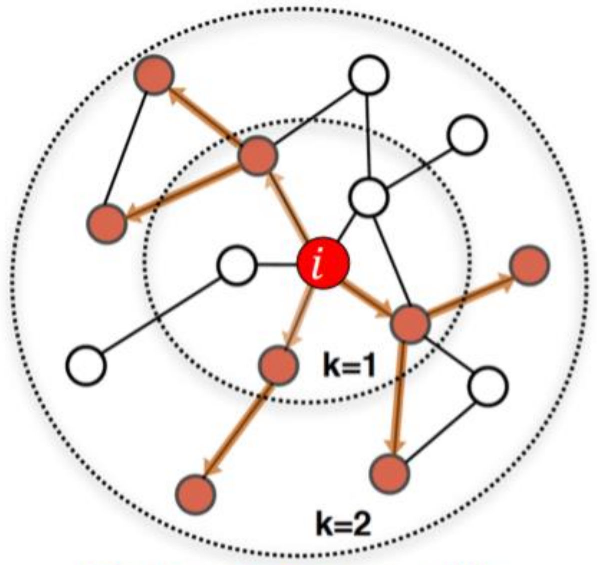
Based on material from:

- Hamilton et al. 2017. Representation Learning on Graphs: Methods and Applications. *IEEE Data Engineering Bulletin on Graph Systems*.
- Scarselli et al. 2005. The Graph Neural Network Model. *IEEE Transactions on Neural Networks*.

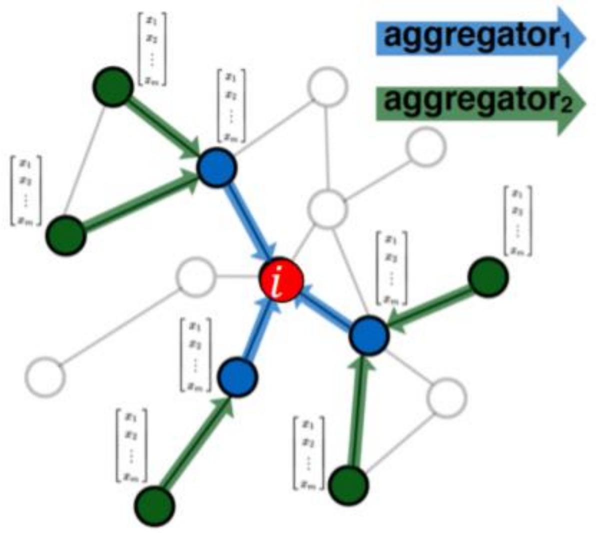
[Kipf and Welling, ICLR 2017]

Graph Convolutional Networks

- Idea: Node's neighborhood defines a computation graph



Determine node computation graph

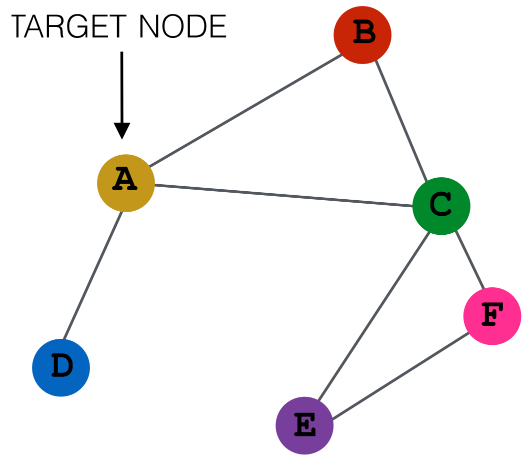


Propagate and transform information

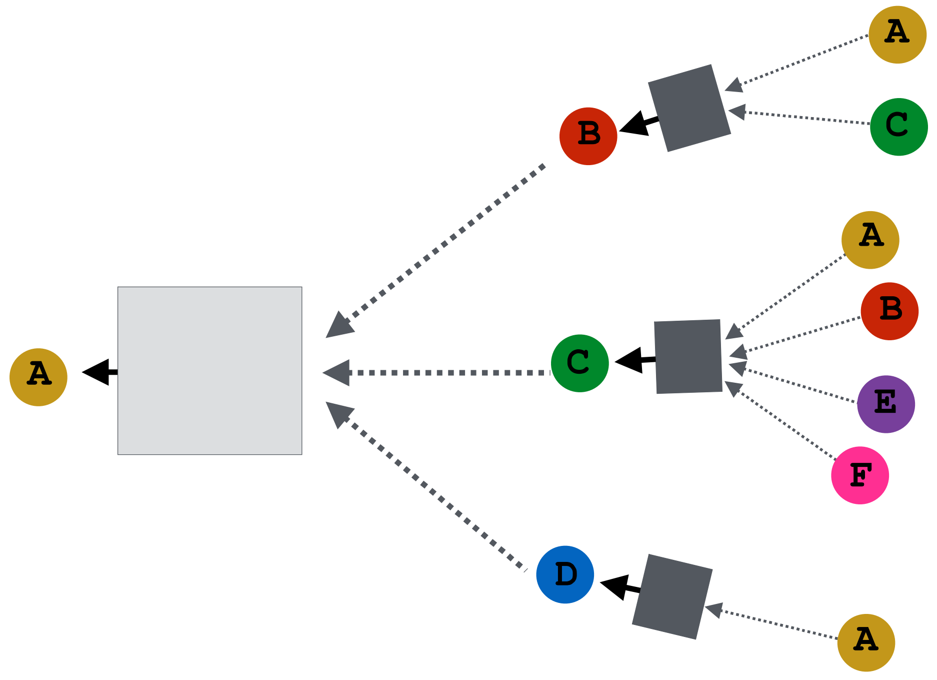
Learn how to propagate information across the graph to compute node features

Neighborhood Aggregation

- **Key idea:** Generate node embeddings based on local neighborhoods.
- **Intuition:** Nodes aggregate information from their neighbors using neural networks

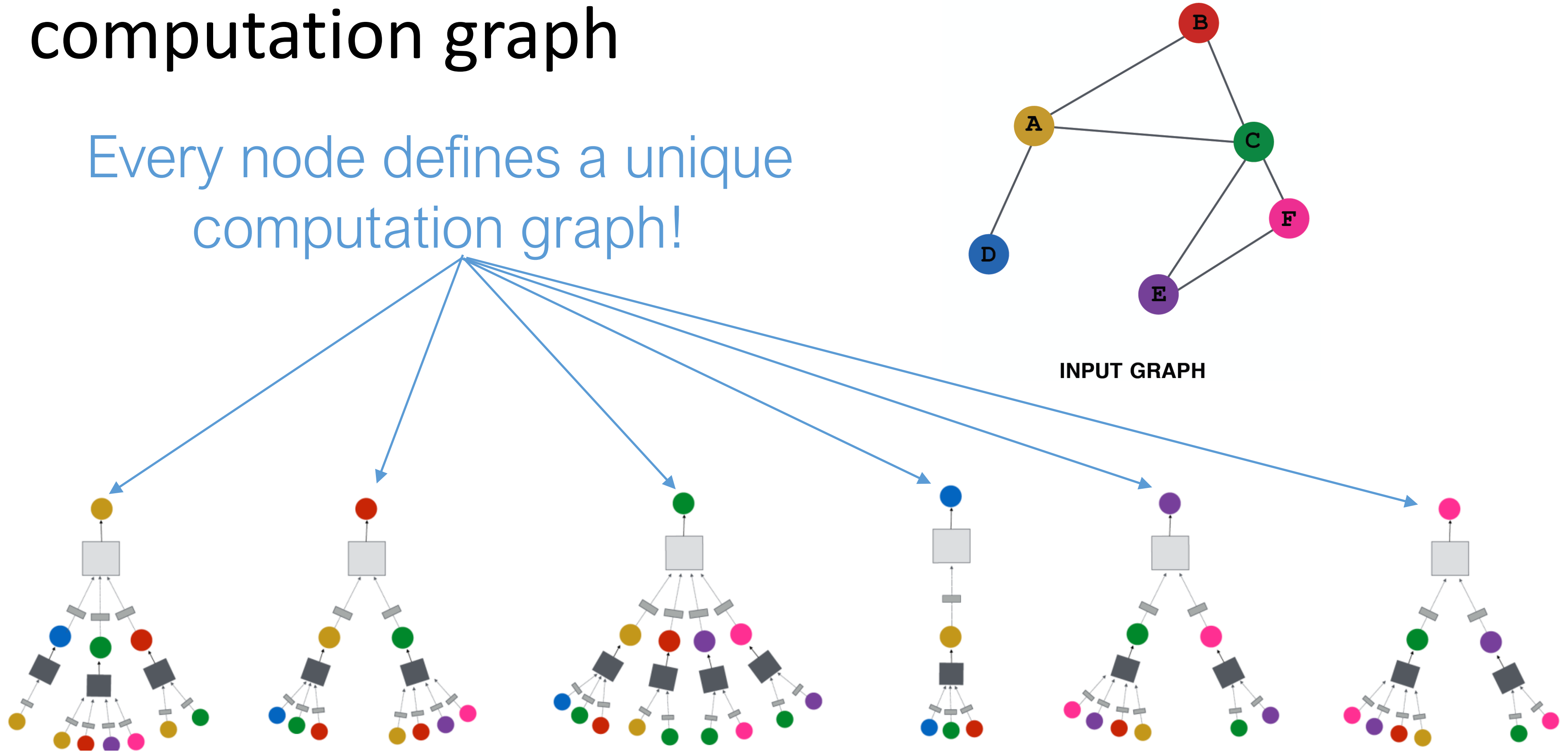


INPUT GRAPH



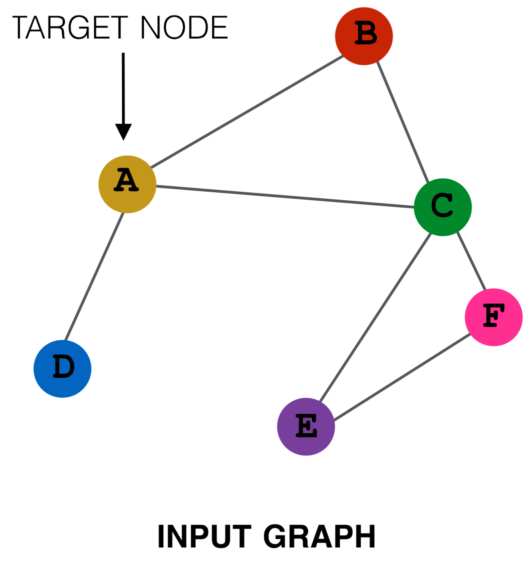
Neighborhood Aggregation

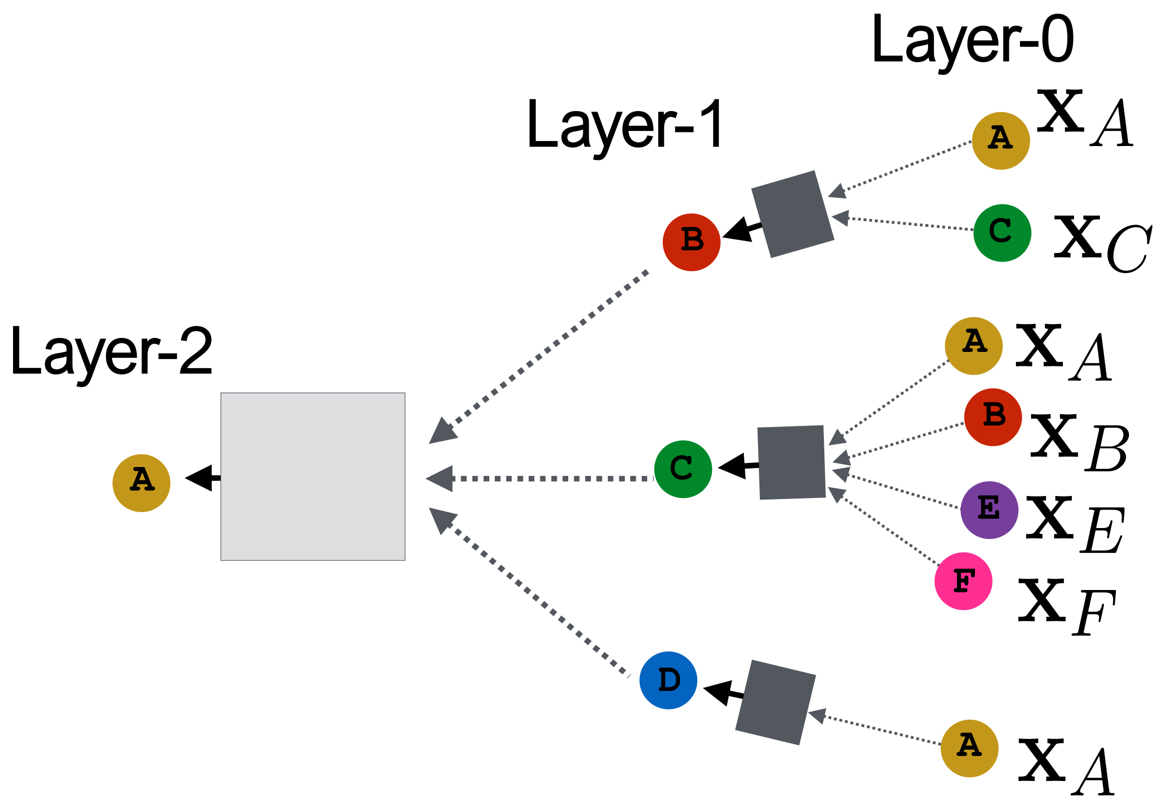
- **Intuition:** Network neighborhood defines a computation graph



Neighborhood Aggregation

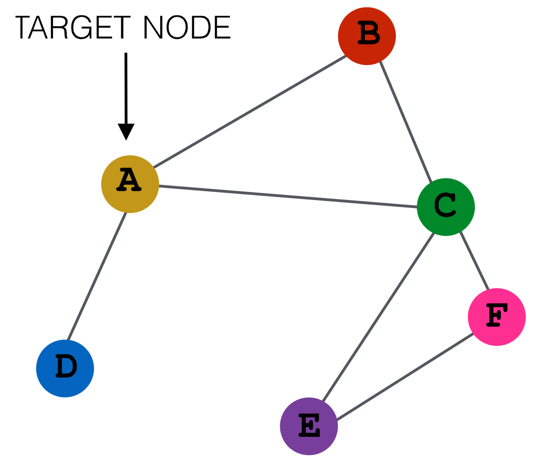
- Nodes have embeddings at each layer.
- Model can be arbitrary depth.
- “Layer-0” embedding of node u is its input feature, i.e. x_u.
- Layer-K embedding gets information from nodes that are K hops away





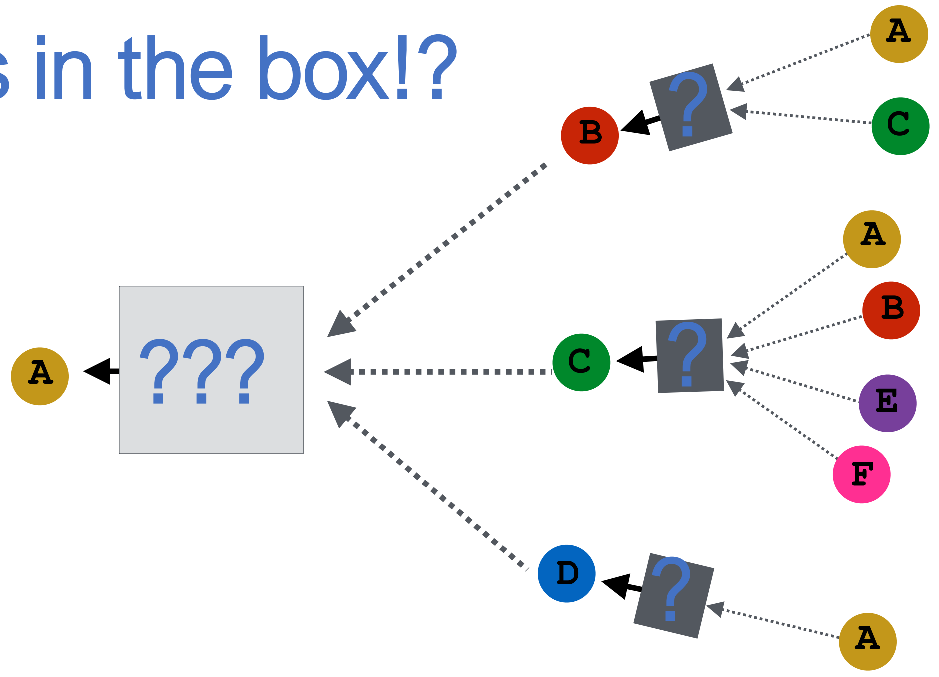
Neighborhood Aggregation

- Key distinctions are in how different approaches aggregate information across the layers.



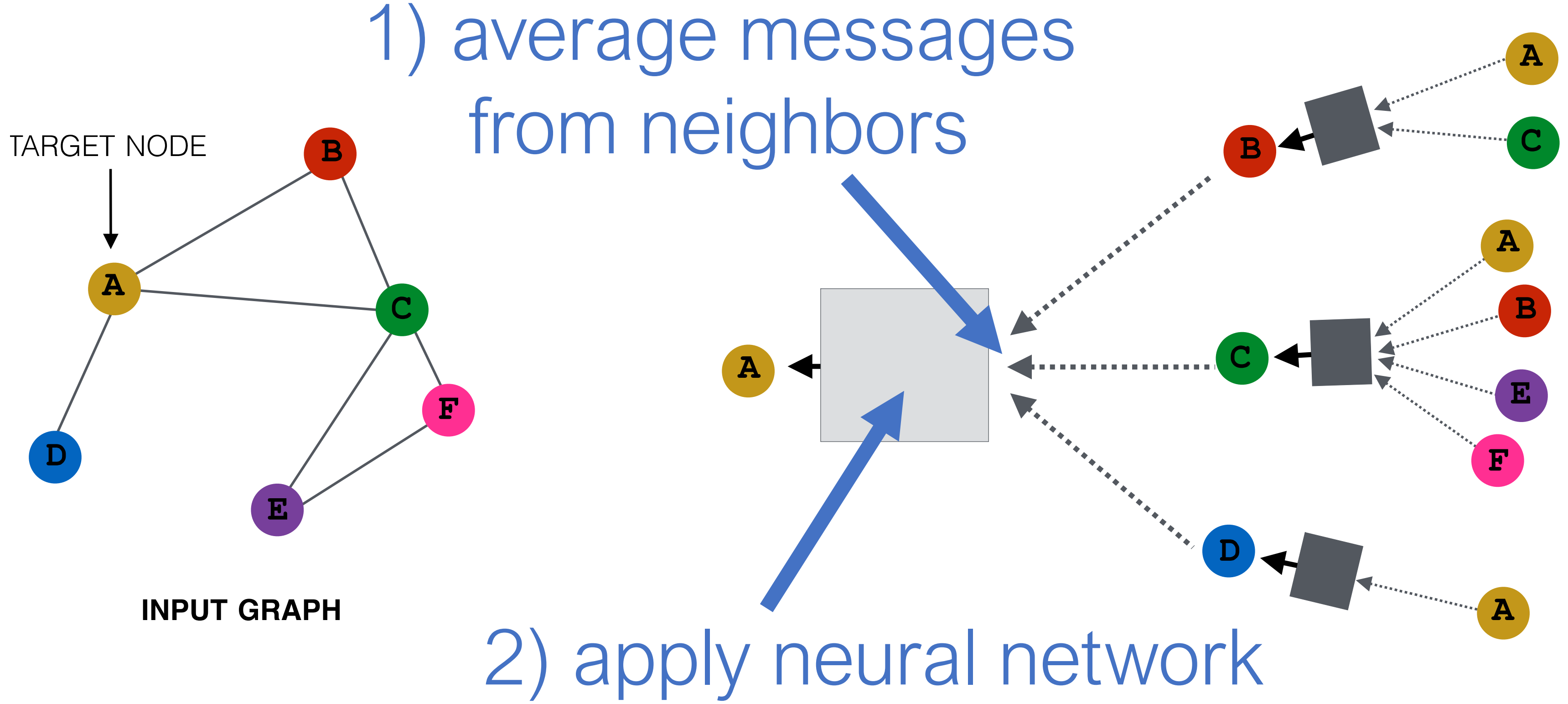
INPUT GRAPH

what's in the box!?



Neighborhood Aggregation

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



The Math

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

Initial "layer 0" embeddings are equal to node features

$$\boxed{\mathbf{h}_v^0 = \mathbf{x}_v}$$

previous layer embedding of v

$$\boxed{\mathbf{h}_v^k} = \boxed{\sigma}\left(\mathbf{W}_k \boxed{\sum_{u \in N(v)} \frac{\mathbf{h}_u^{k-1}}{|N(v)|}} + \mathbf{B}_k \boxed{\mathbf{h}_v^{k-1}}\right), \quad \forall k > 0$$

kth layer embedding of v

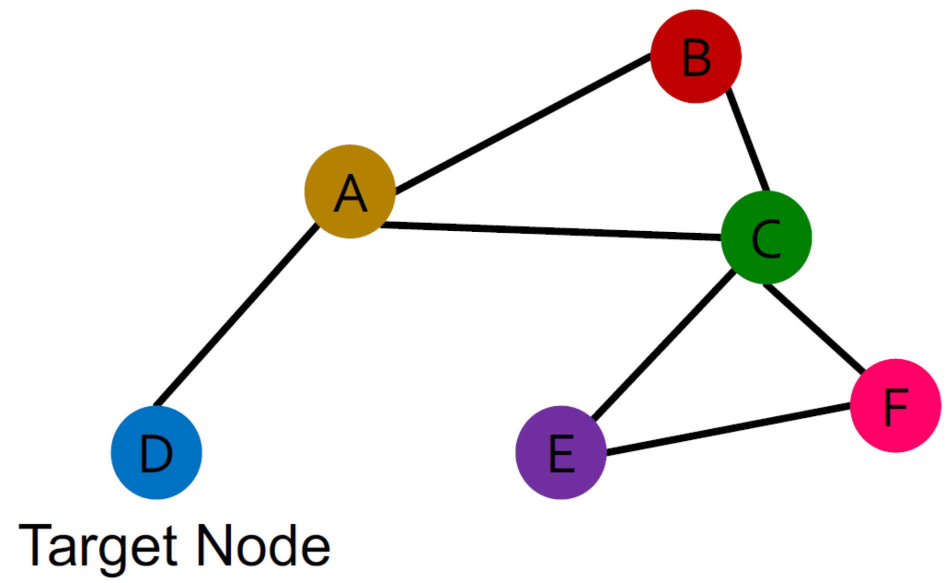
non-linearity (e.g., ReLU or tanh)

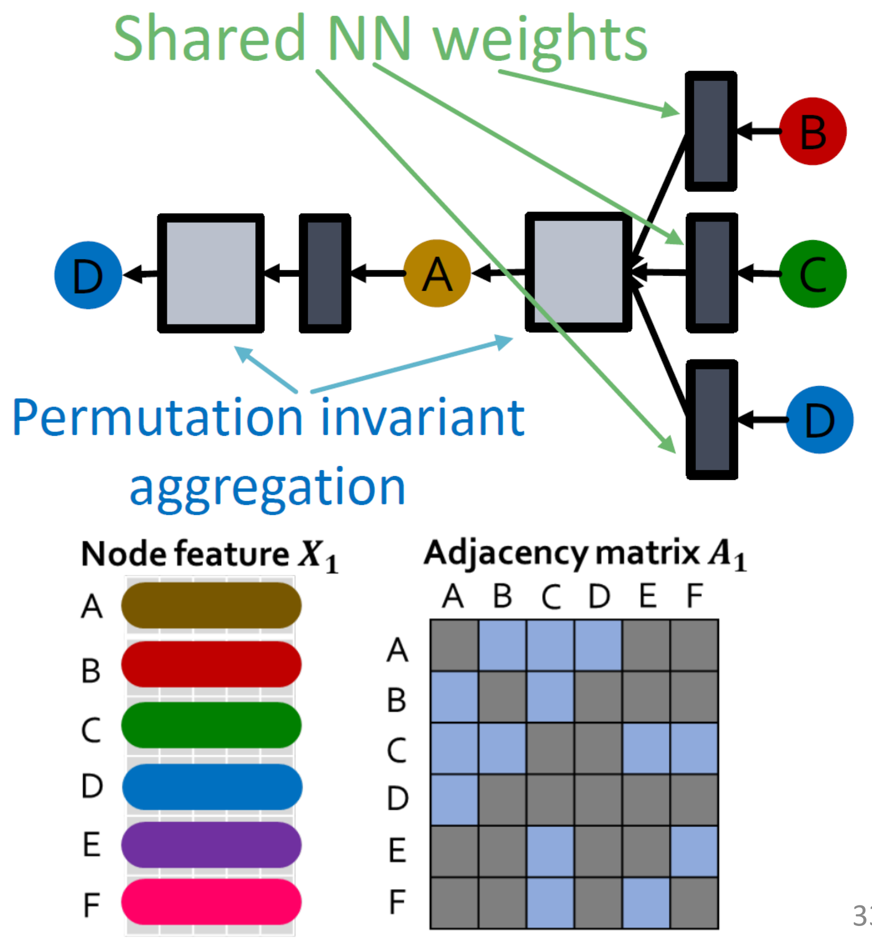
average of neighbor's previous layer embeddings

Notice summation is a permutation invariant pooling/aggregation.

Equivariant Property

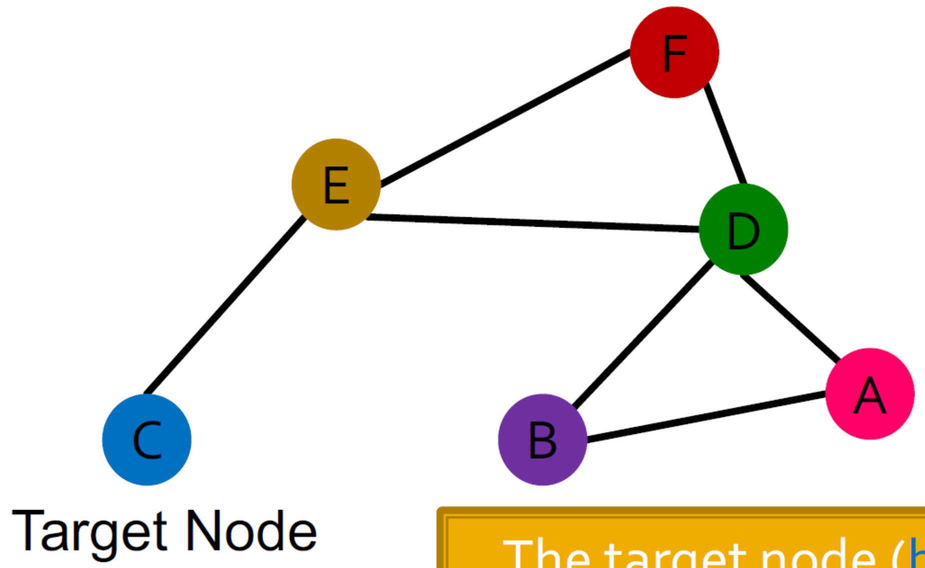
- **Message passing and neighbor aggregation in graph convolution networks is permutation equivariant**



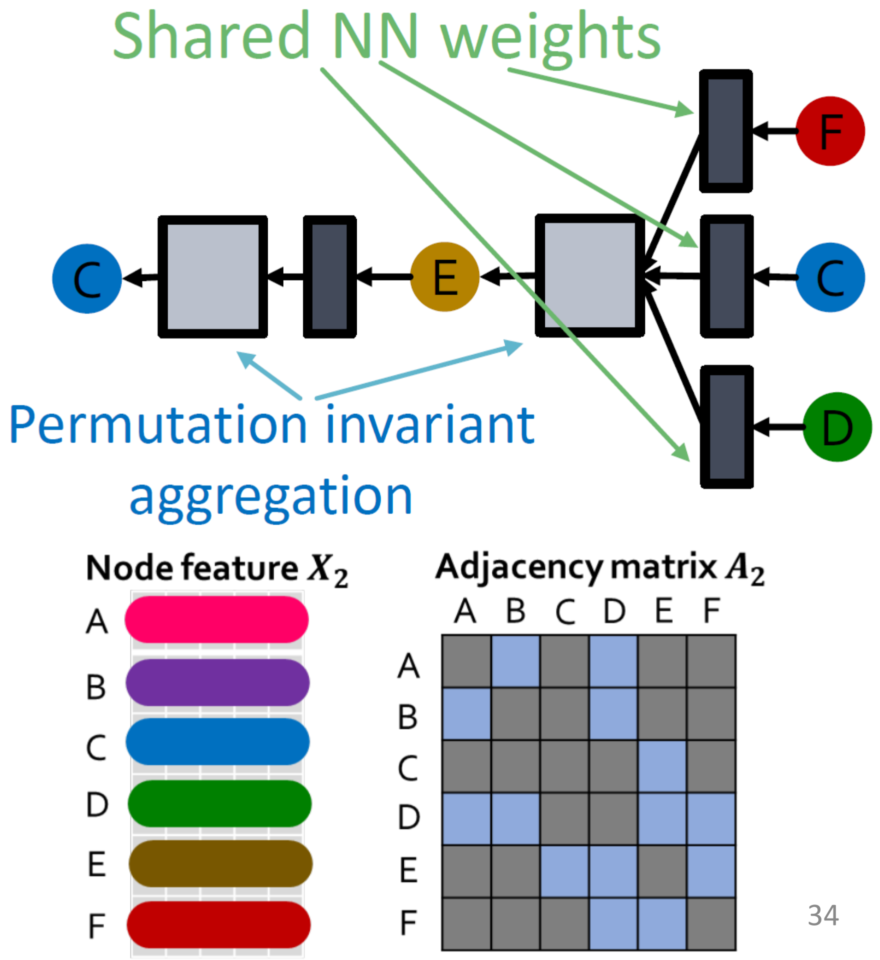


Equivariant Property

- **Message passing and neighbor aggregation in graph convolution networks is permutation equivariant**



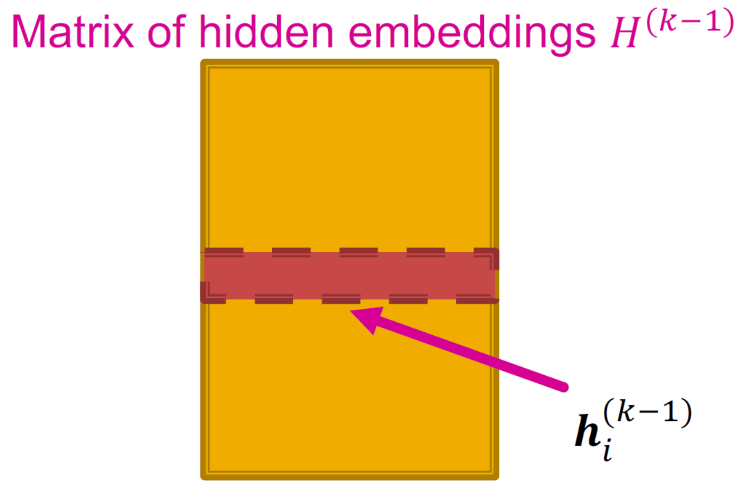
The target node (blue) has the same computation graph for different order plans



Matrix Formulation

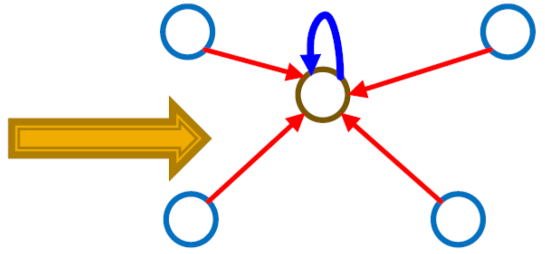
- **Many aggregations can be performed efficiently by (sparse) matrix operations**
  - Let $H^k = \left[h_1^k, \ldots, h_V^k\right]^T$
  - Then $\sum_{u \in N_v} h_u^k = A_{v,:} H^k$
    - N_v is the set of neighbours of *v*
  - Let D be diagonal matrix where
    - $D_{v,v} = Degree(v) = |N_v|$
    - The inverse of D is also a diagonal matrix
      - $D_{v,v}^{-1} = 1/|N_v|$
  - Therefore

$$\sum_{u \in N_v} \frac{h_u^k}{|N_v|} \Longrightarrow H^{k+1} = D^{-1} A H^k$$

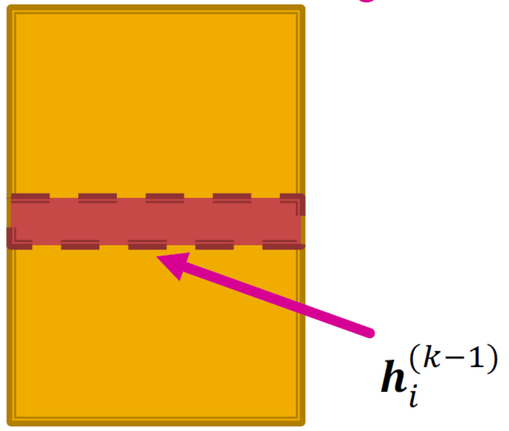

Matrix of hidden embeddings $H^{(k-1)}$
$\boldsymbol{h}_i^{(k-1)}$

Matrix Formulation

- Re-writing update function in matrix form
$$H^{k+1} = \sigma(D^{-1}AH^kW_k^T + H^kB_k^T)$$
  - Red: neighbourhood aggregation
  - Blue: self transformation


$$H^{(k)} = [h_1^{(k)} \dots h_{|V|}^{(k)}]^T$$

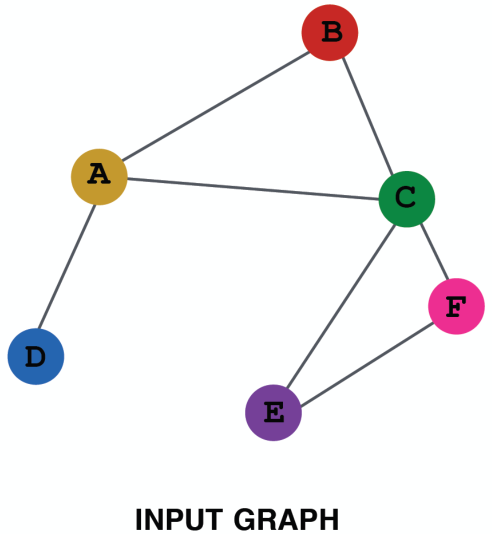
- In practice, this implies that efficient sparse matrix multiplication can be used (*A* is sparse)

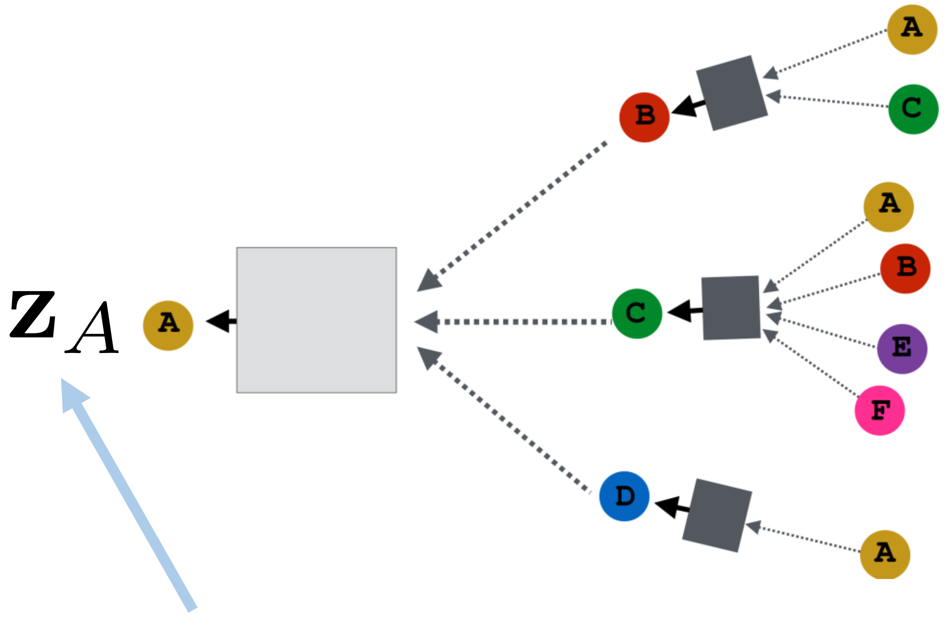
Matrix of hidden embeddings $H^{(k-1)}$


- Note: not all GNNs can be expressed in matrix form, when aggregation function is complex

Training the Model

- How do we train the model to generate "high-quality" embeddings?





Need to define a loss function on the embeddings, $L(z_A)$!

Training the Model

$\mathbf{W}_k$: weight matrix for neighborhood aggregation
$\mathbf{B}_k$: weight matrix for transforming hidden vector of self

trainable matrices
(i.e., what we learn)

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

$$\mathbf{h}_v^k = \sigma\left(\boxed{\mathbf{W}_k} \sum_{u \in N(v)} \frac{\mathbf{h}_u^{k-1}}{|N(v)|} + \boxed{\mathbf{B}_k} \mathbf{h}_v^{k-1}\right), \quad \forall k \in \{1, ..., K\}$$

$$\boxed{\mathbf{z}_v = \mathbf{h}_v^K}$$

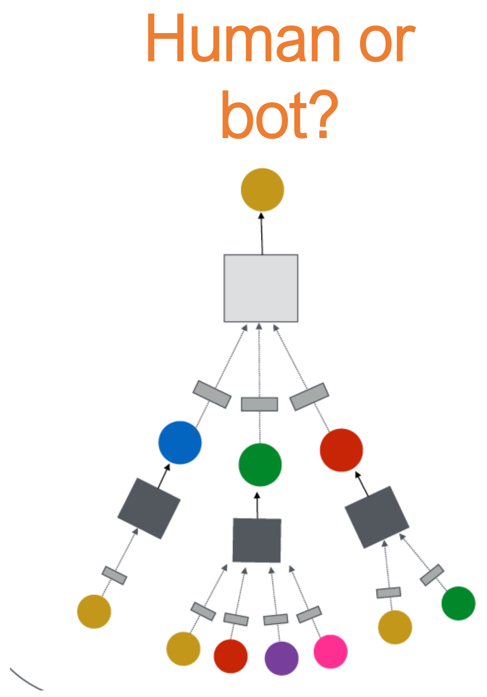
- After K-layers of neighborhood aggregation, we get output embeddings for each node.
- **We can feed these embeddings into any loss function** and run stochastic gradient descent to train the aggregation parameters.

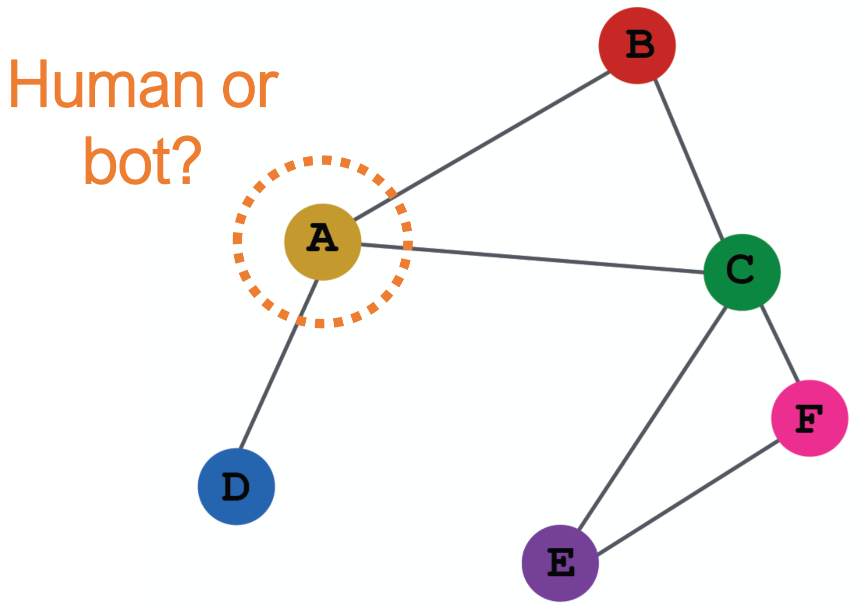
Training the Model

- Train in an **unsupervised manner** using only the graph structure.
  - Use only the graph structure
  - “Similar” nodes have similar embeddings
- Unsupervised loss function can be anything from the last section, e.g., based on
  - Random walks (node2vec, DeepWalk)
  - Adjacent neighbor similarity
  - Node proximity in the graph (higher order relation similarity)

Training the Model

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





e.g., an online social network

Training the Model

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

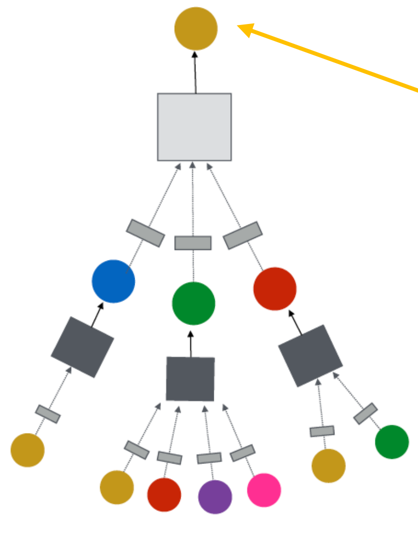
Human or bot?

$$\mathcal{L} = \sum_{v \in V} \boxed{y_v} \log(\sigma(\boxed{\mathbf{z}_v}^\top \boxed{\theta})) + (1 - \boxed{y_v}) \log(1 - \sigma(\boxed{\mathbf{z}_v}^\top \boxed{\theta}))$$

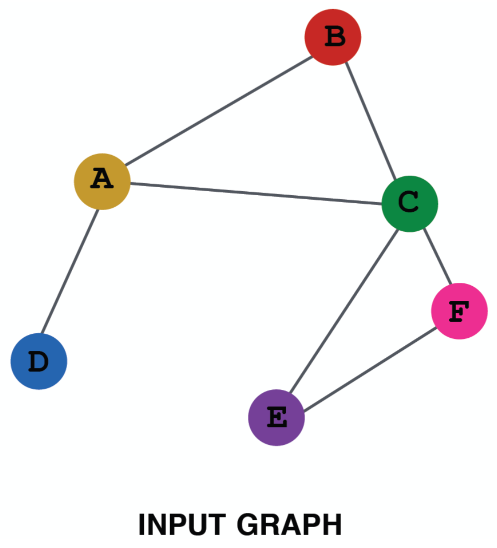
classification weights

output node embedding

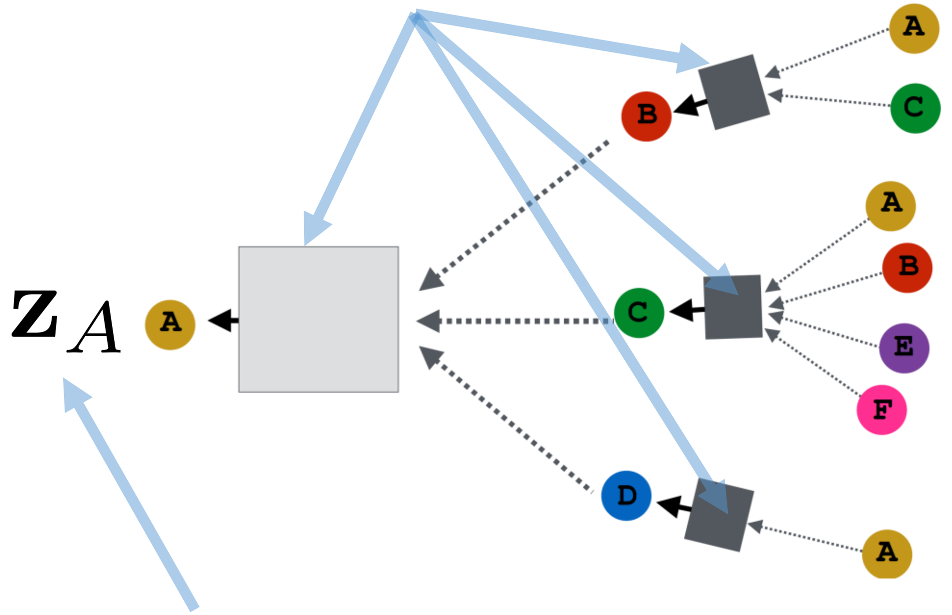
node class label



Overview of Model Design

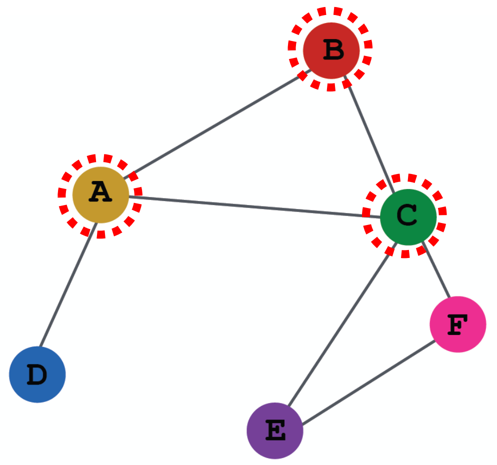


1) Define a neighborhood aggregation function.



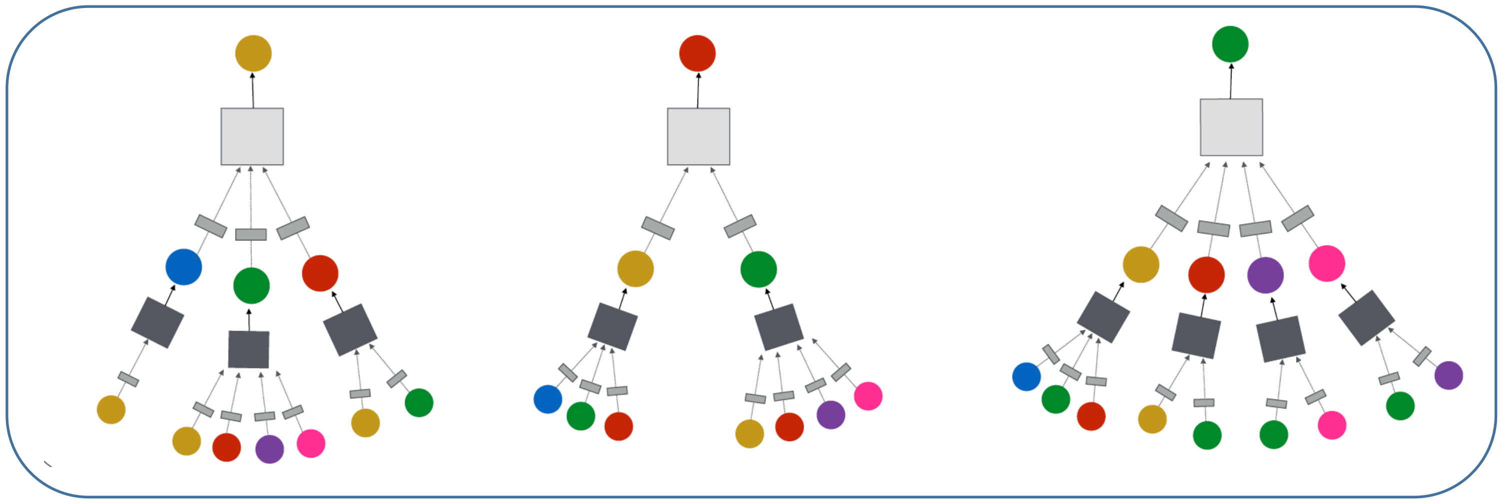
2) Define a loss function on the embeddings, $\mathtt{L(z_u)}$

Overview of Model Design

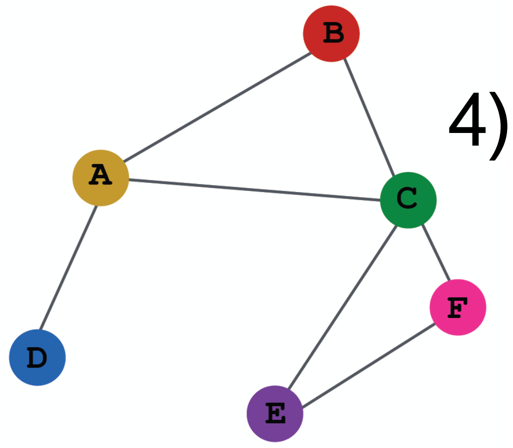


INPUT GRAPH

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



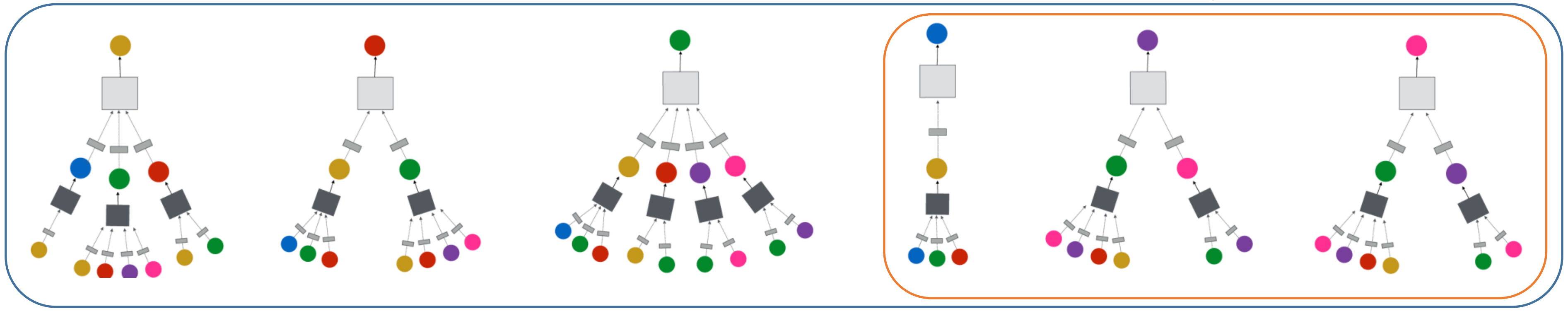
Overview of Model



INPUT GRAPH

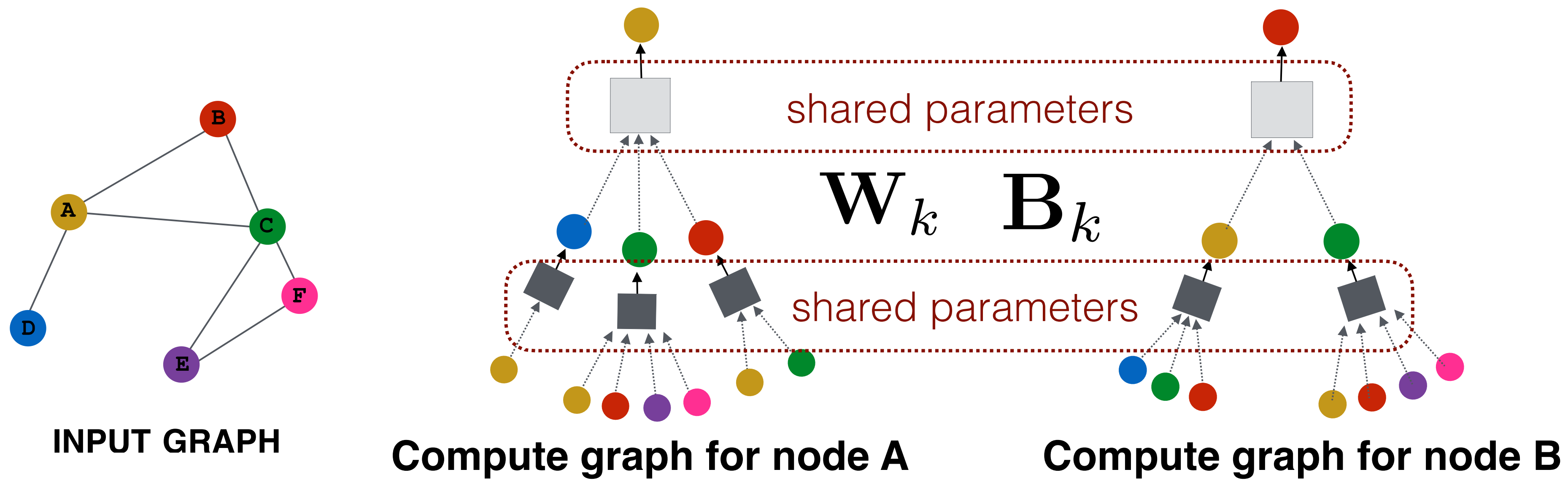
4) Generate embeddings for nodes as needed

Even for nodes we never trained on!!!!

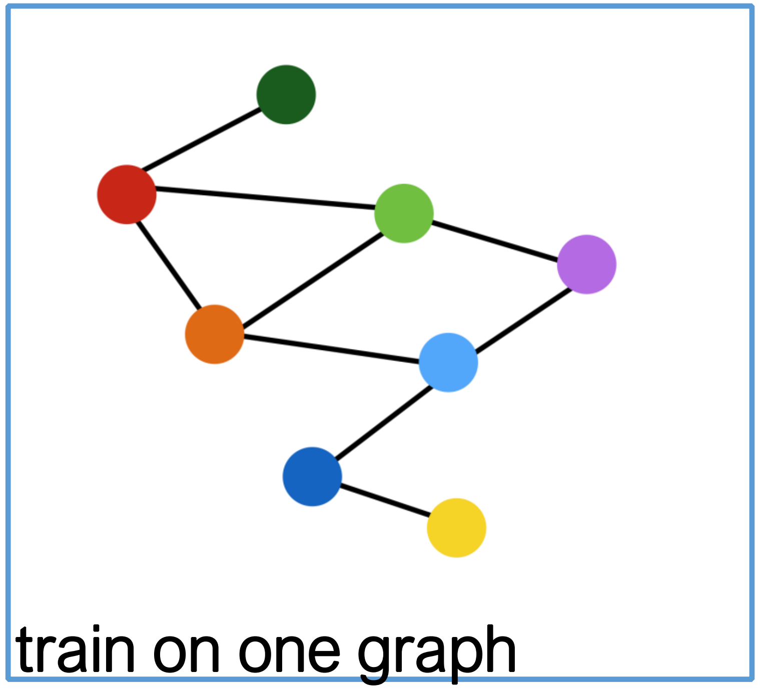


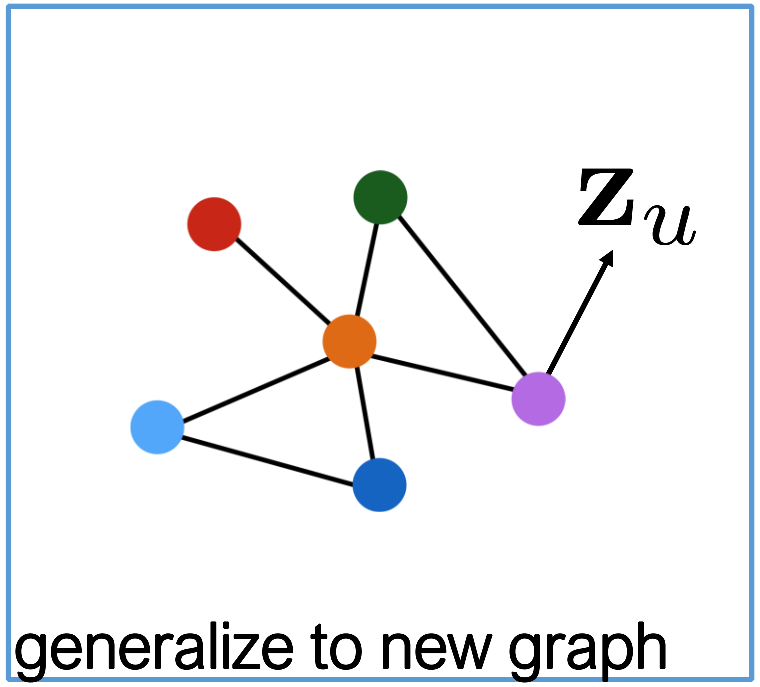
Inductive Capability

- The same aggregation parameters are shared for all nodes.



Inductive Capability: New Graphs

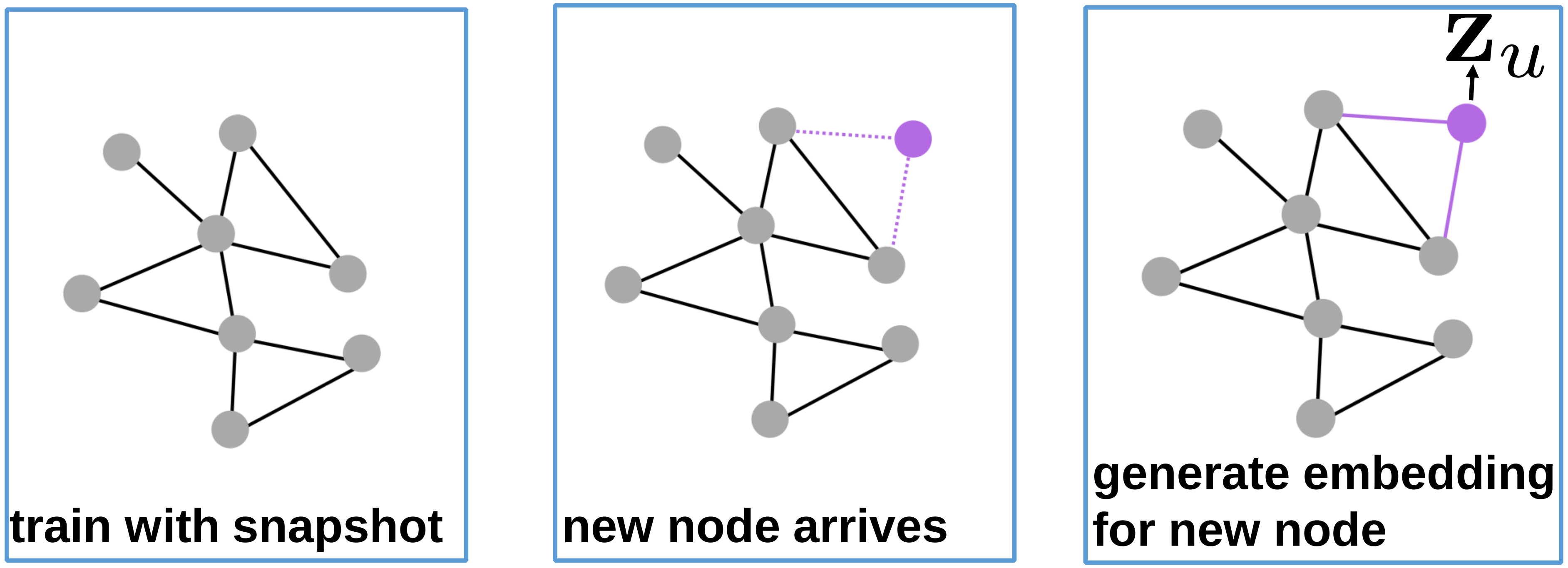




Inductive node embedding $\rightarrow$ generalize to entirely unseen graphs

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

Inductive Capability



Many application settings constantly encounter previously unseen nodes.

e.g., Reddit, YouTube, GoogleScholar, ….

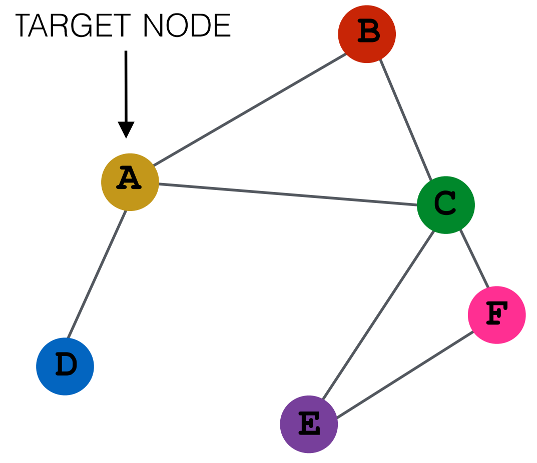
Need to generate new embeddings “on the fly”

Quick Recap

- **Recap:** Generate node embeddings by aggregating neighborhood information.
  - Allows for parameter sharing in the encoder.
  - Allows for inductive learning.

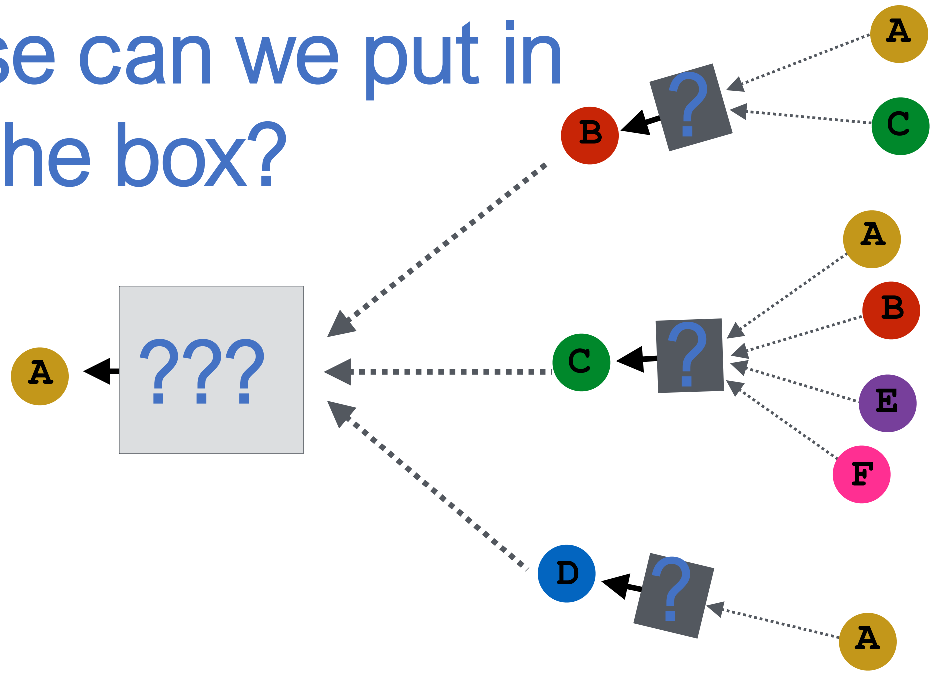
Neighborhood Aggregation

- Key distinctions are in how different approaches aggregate messages



INPUT GRAPH

What else can we put in the box?



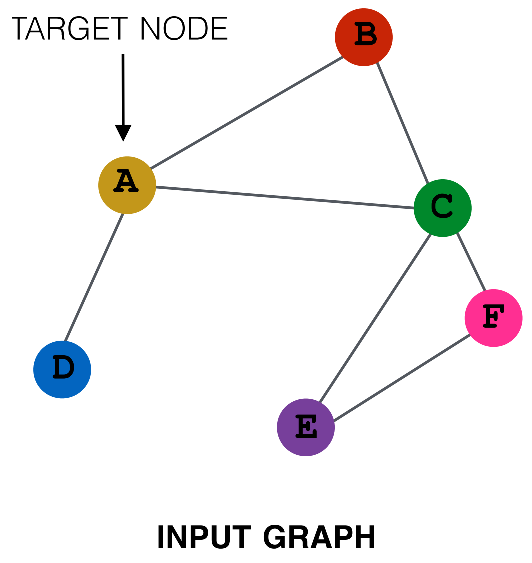
GraphSAGE

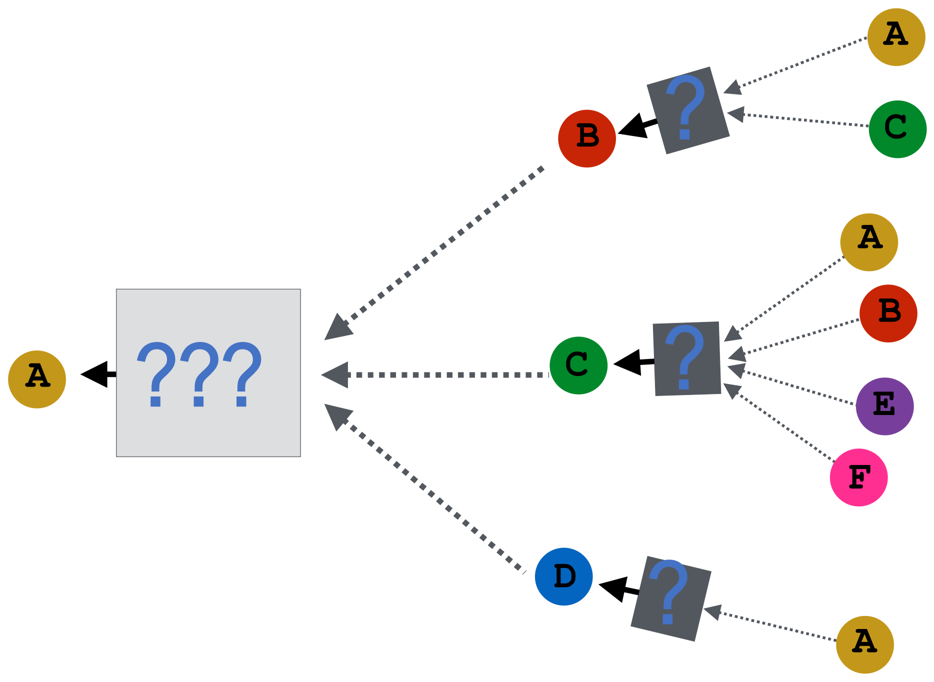
Based on material from:

- Hamilton et al., 2017. Inductive Representation Learning on Large Graphs. *NIPS.*

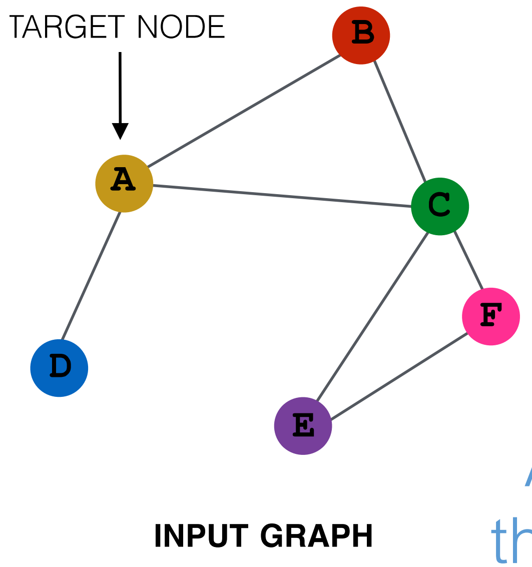
GraphSAGE Idea

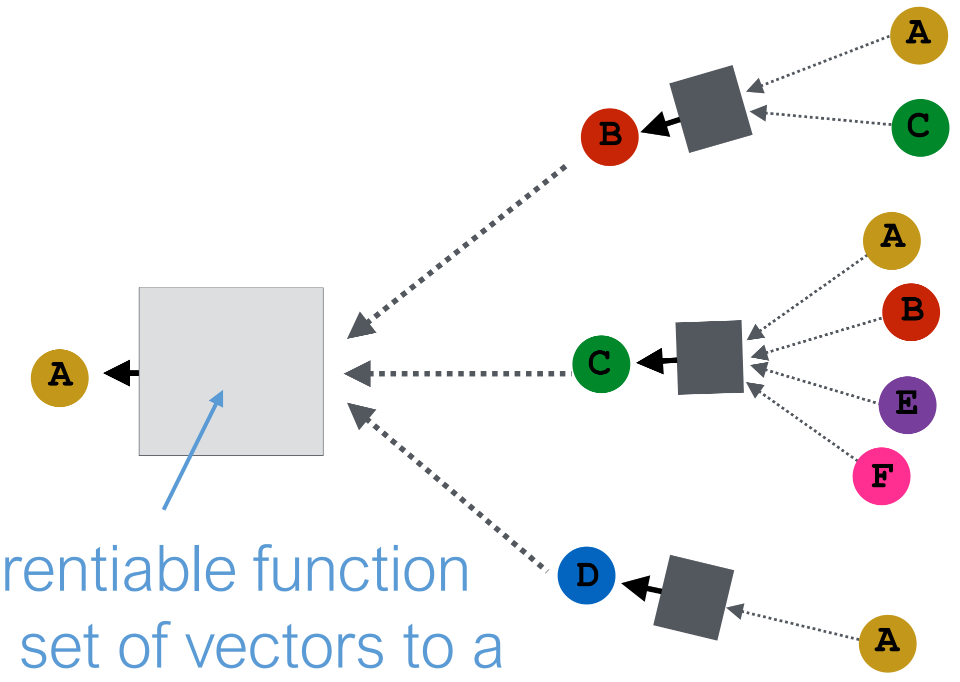
- So far we have aggregated the neighbor messages by taking their (weighted) average, can we do better?





GraphSAGE Idea





Any differentiable function that maps set of vectors to a single vector.

$$\mathbf{h}_v^k = \sigma\left(\left[\mathbf{A}_k \cdot \boxed{\text{AGG}(\{\mathbf{h}_u^{k-1}, \forall u \in N(v)\})}, \mathbf{B}_k \mathbf{h}_v^{k-1}\right]\right)$$

Apply L2 normalization for each node embedding at every layer

GraphSAGE Differences

- Simple neighborhood aggregation:

$$\mathbf{h}_v^k = \sigma\left(\mathbf{W}_k \sum_{u \in N(v)} \frac{\mathbf{h}_u^{k-1}}{|N(v)|} + \mathbf{B}_k \mathbf{h}_v^{k-1}\right)$$

- GraphSAGE:

concatenate self embedding and neighbor embedding

$$\mathbf{h}_v^k = \sigma\left(\left[\mathbf{W}_k \cdot \boxed{\text{AGG}\left(\{\mathbf{h}_u^{k-1}, \forall u \in N(v)\}\right)}, \mathbf{B}_k \mathbf{h}_v^{k-1}\right]\right)$$

generalized aggregation

GraphSAGE Variants

- **Mean:**

$$\text{AGG} = \sum_{u \in N(v)} \frac{\mathbf{h}_u^{k-1}}{|N(v)|}$$

- **Pool**
  - Transform neighbor vectors and apply symmetric vector function.

element-wise mean/max

$$\text{AGG} = \boxed{\gamma}\left(\{\mathbf{Q}\mathbf{h}_u^{k-1}, \forall u \in N(v)\}\right)$$

- **LSTM:**
  - Apply LSTM to random permutation of neighbors.

$$\text{AGG} = \text{LSTM}\left([\mathbf{h}_u^{k-1}, \forall u \in \pi(N(v))]\right)$$

Summary

- **The Basics**
- **GraphSAGE**
  - Generalized neighborhood aggregation.

- **Still a very active research field**