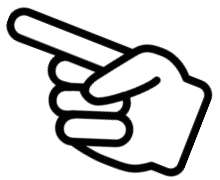
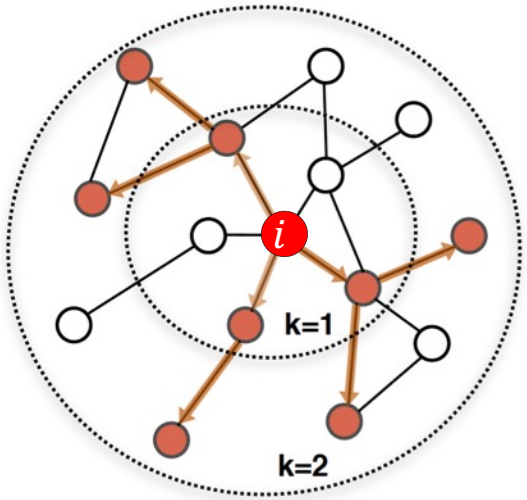
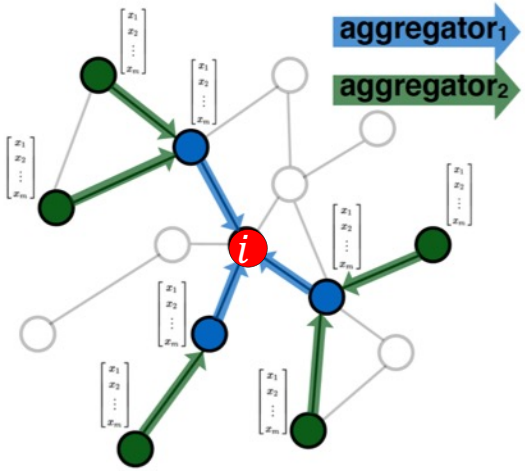
1. Basics of deep learning ✓
2. Deep learning for graphs ✓
3. **Graph Convolutional Networks** 
4. **GNNs subsume CNNs and Transformers**

Graph Convolutional Networks

Idea: Node's neighborhood defines a computation graph



Determine node computation graph

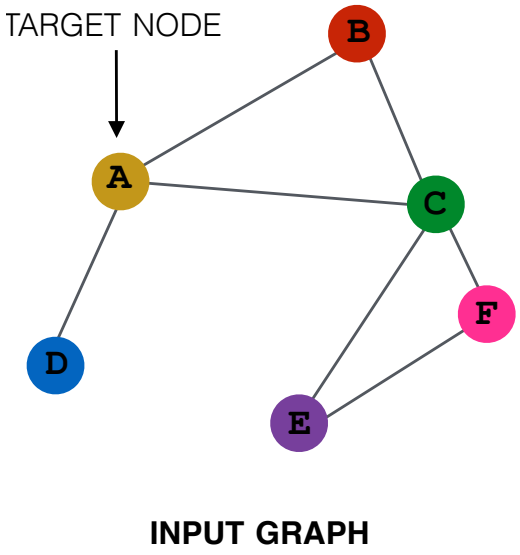


Propagate and transform information

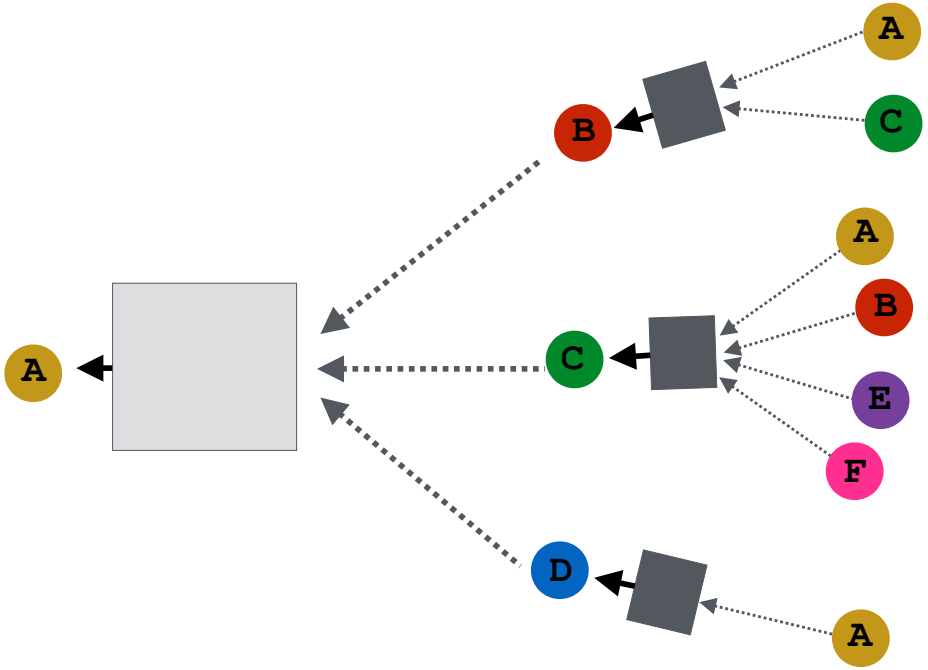
Learn how to propagate information across the graph to compute node features

Idea: Aggregate Neighbors

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

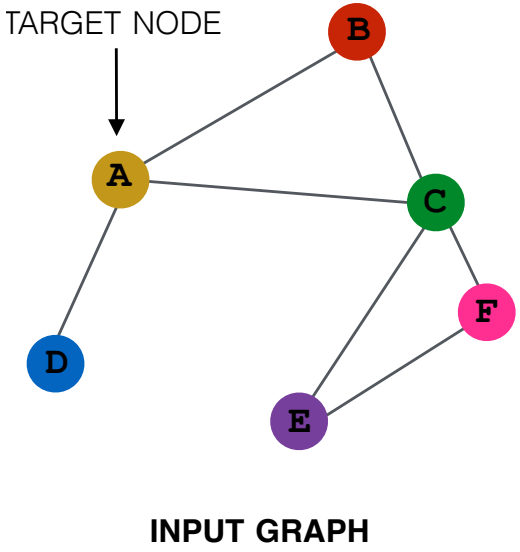


INPUT GRAPH

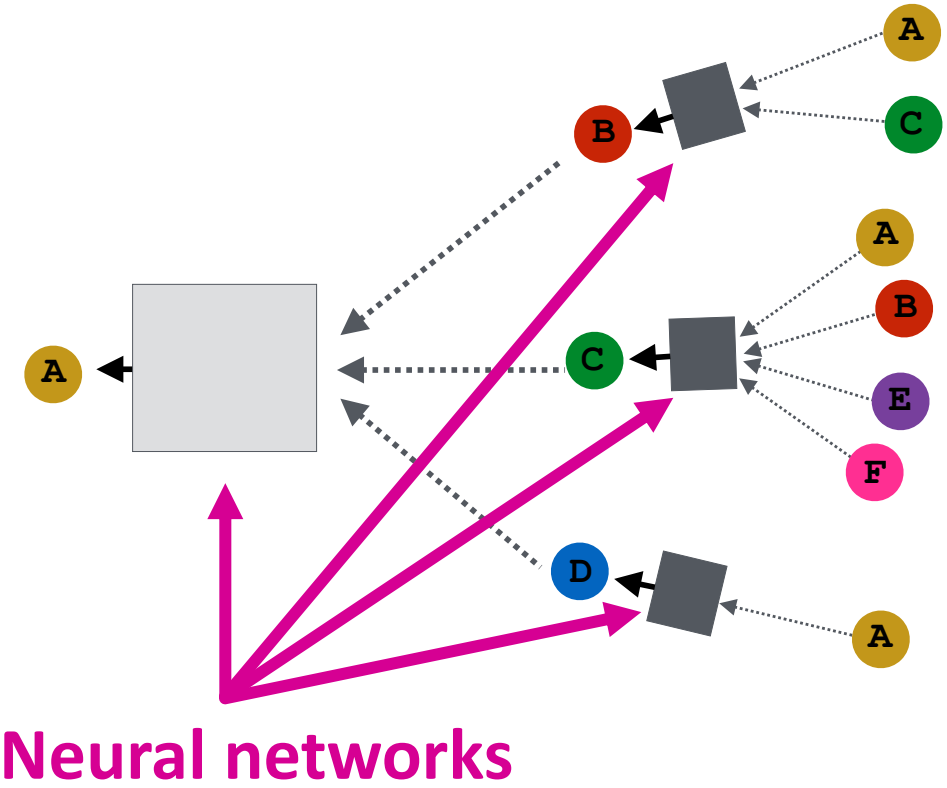


Idea: Aggregate Neighbors

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



INPUT GRAPH

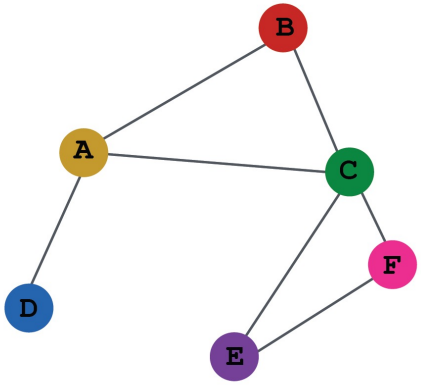


Neural networks

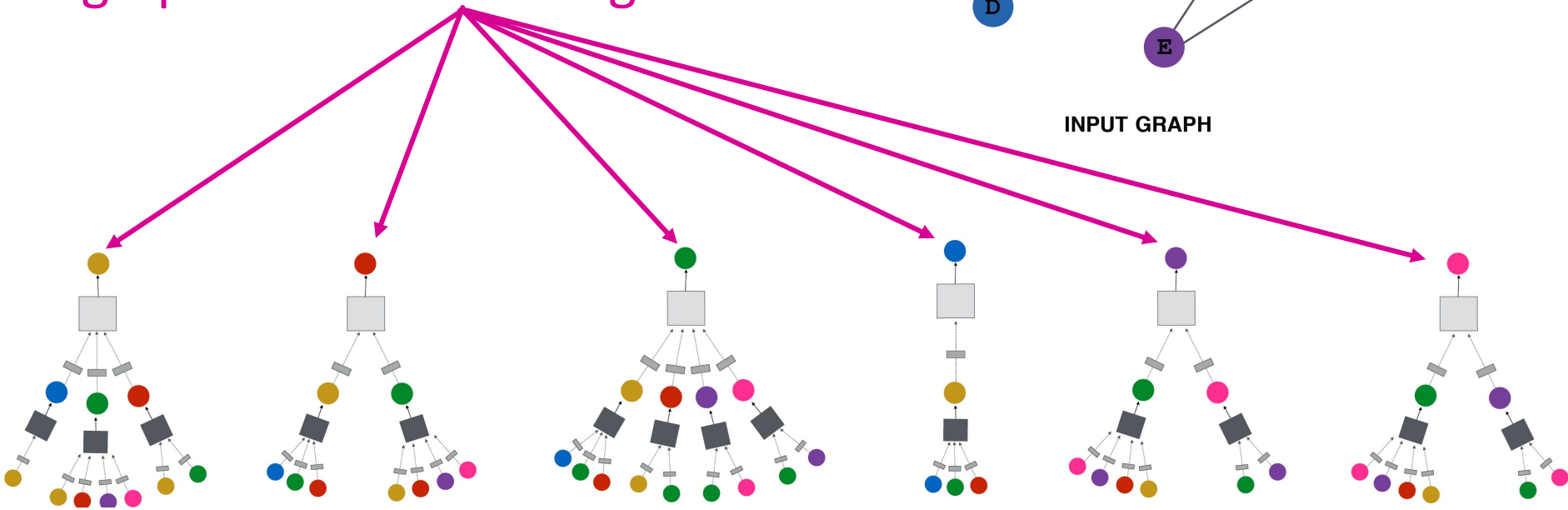
Idea: Aggregate Neighbors

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

Every node defines a computation graph based on its neighborhood!

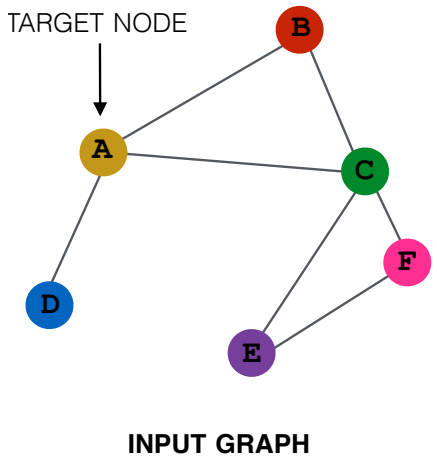


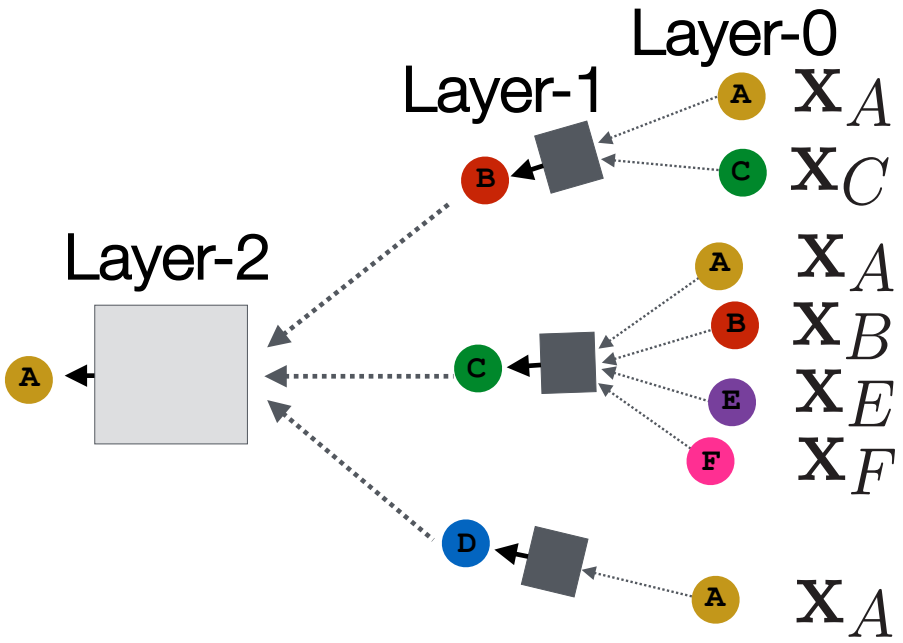
INPUT GRAPH



Deep Model: Many Layers

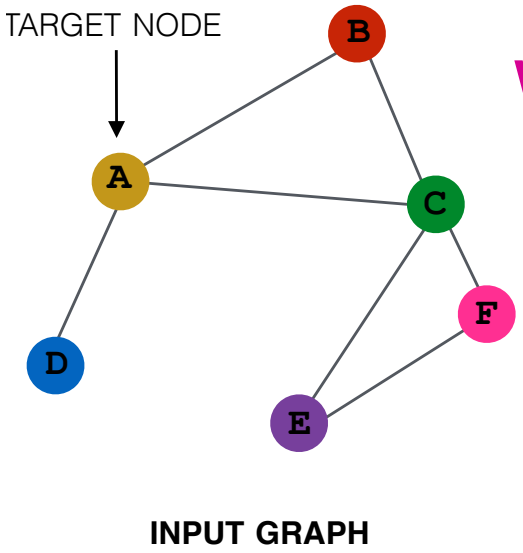
- Model can be of arbitrary depth:
  - Nodes have embeddings at each layer
  - Layer-0 embedding of node v is its input feature, x_v
  - Layer-k embedding gets information from nodes that are k hops away



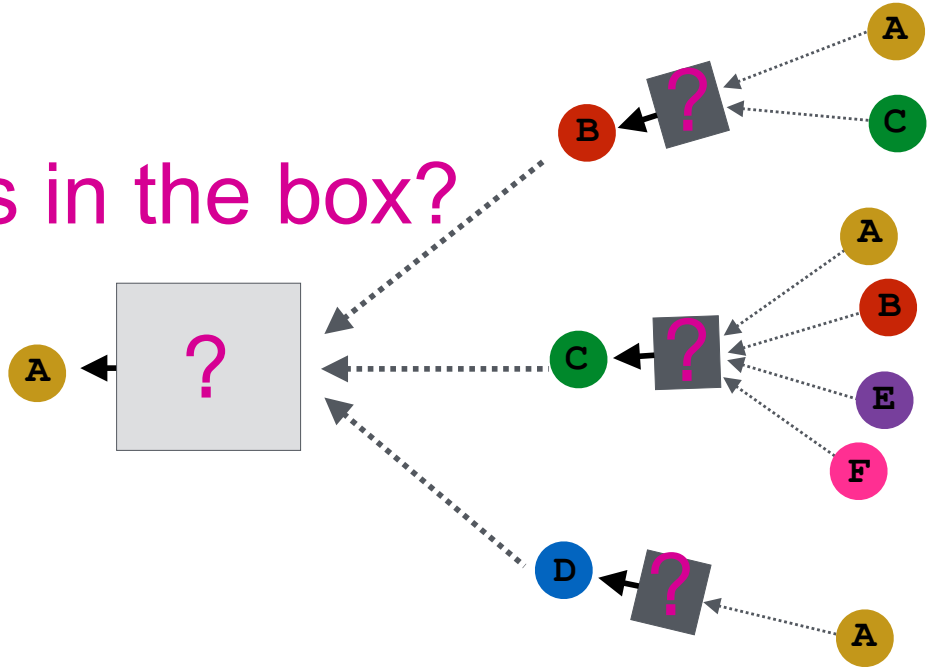


Neighborhood Aggregation

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

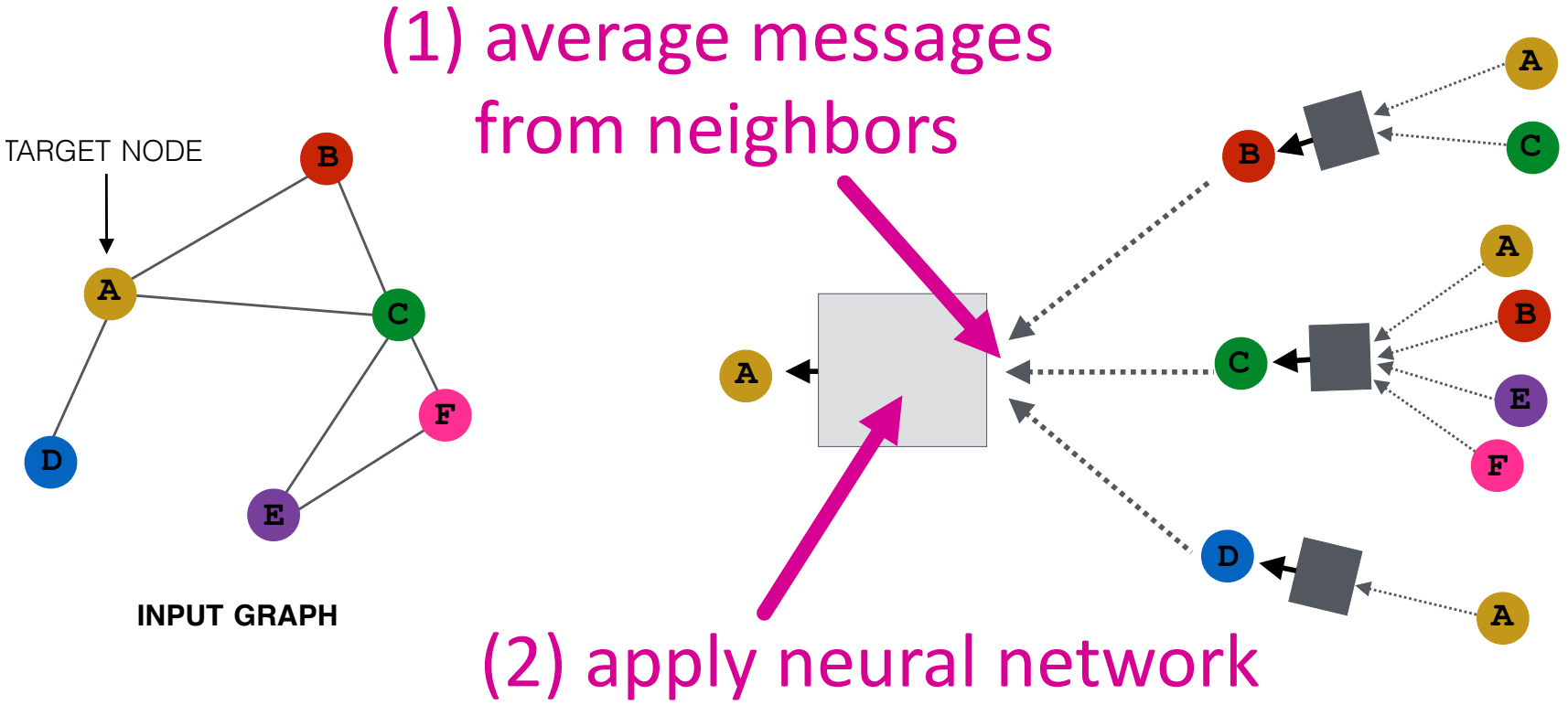


What is in the box?



Neighborhood Aggregation

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



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 k

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

Total number of layers

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

Embedding after L layers of neighborhood aggregation

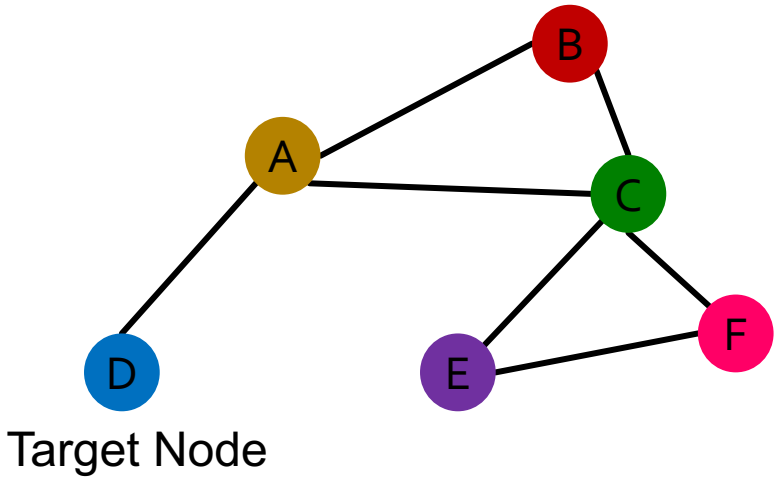
Non-linearity (e.g., ReLU)

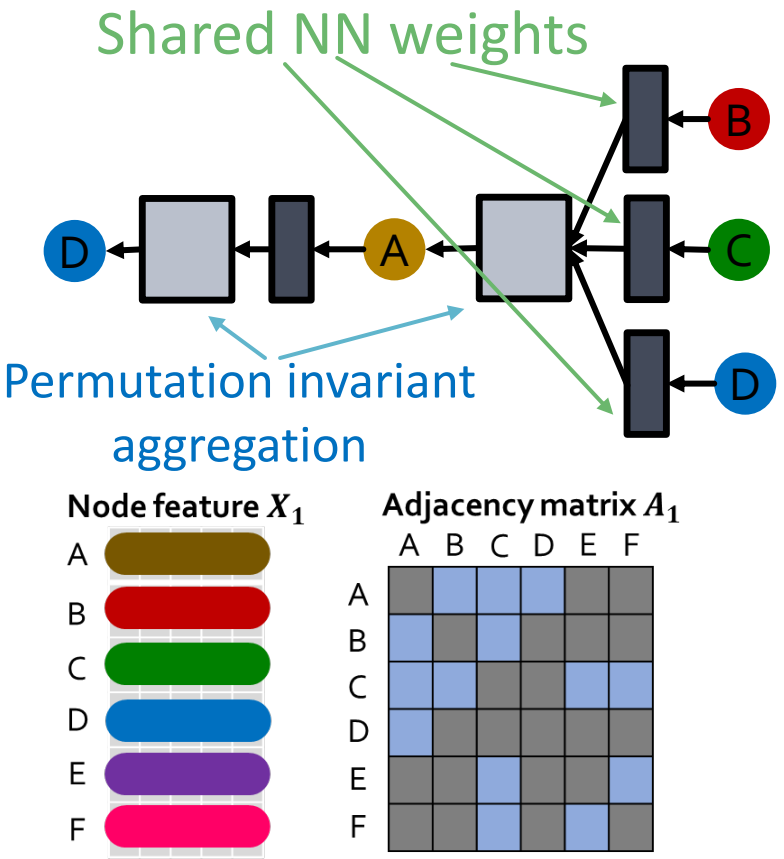
Average of neighbor's previous layer embeddings

Notice summation is a permutation invariant pooling/aggregation.

Equivariant Property

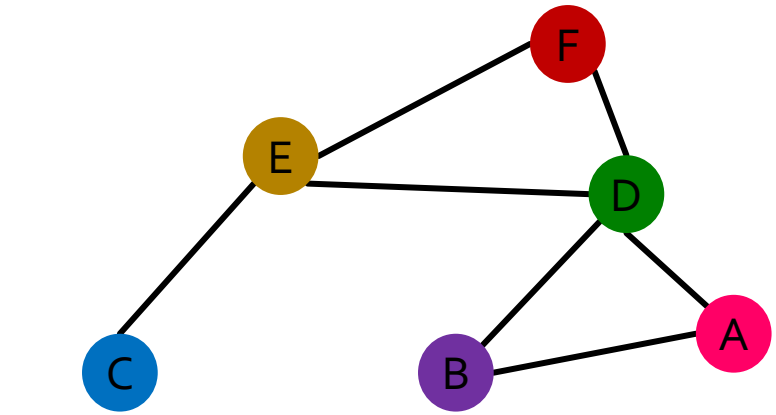
Message passing and neighbor aggregation in graph convolution networks is permutation equivariant.





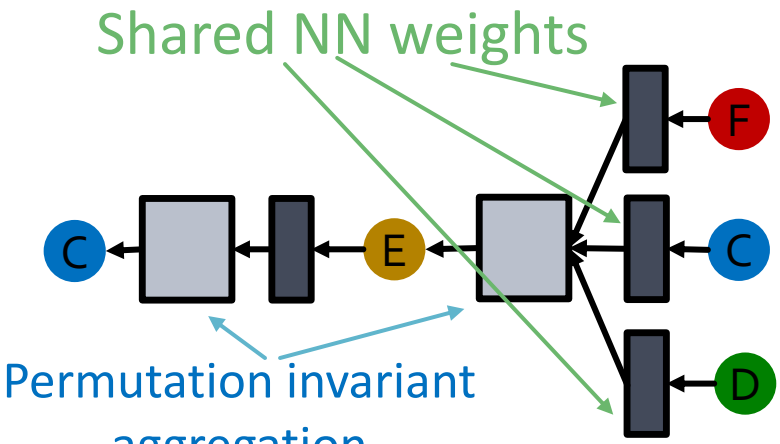
Equivariant Property

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

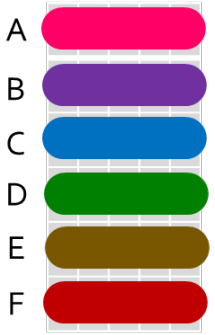


Target Node

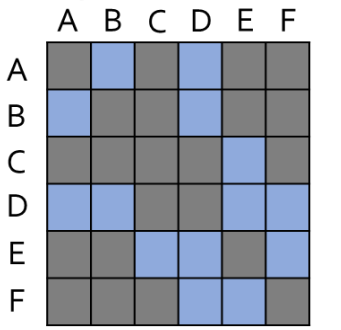
The target node (blue) has the same computation graph for different order plans



Node feature X_2

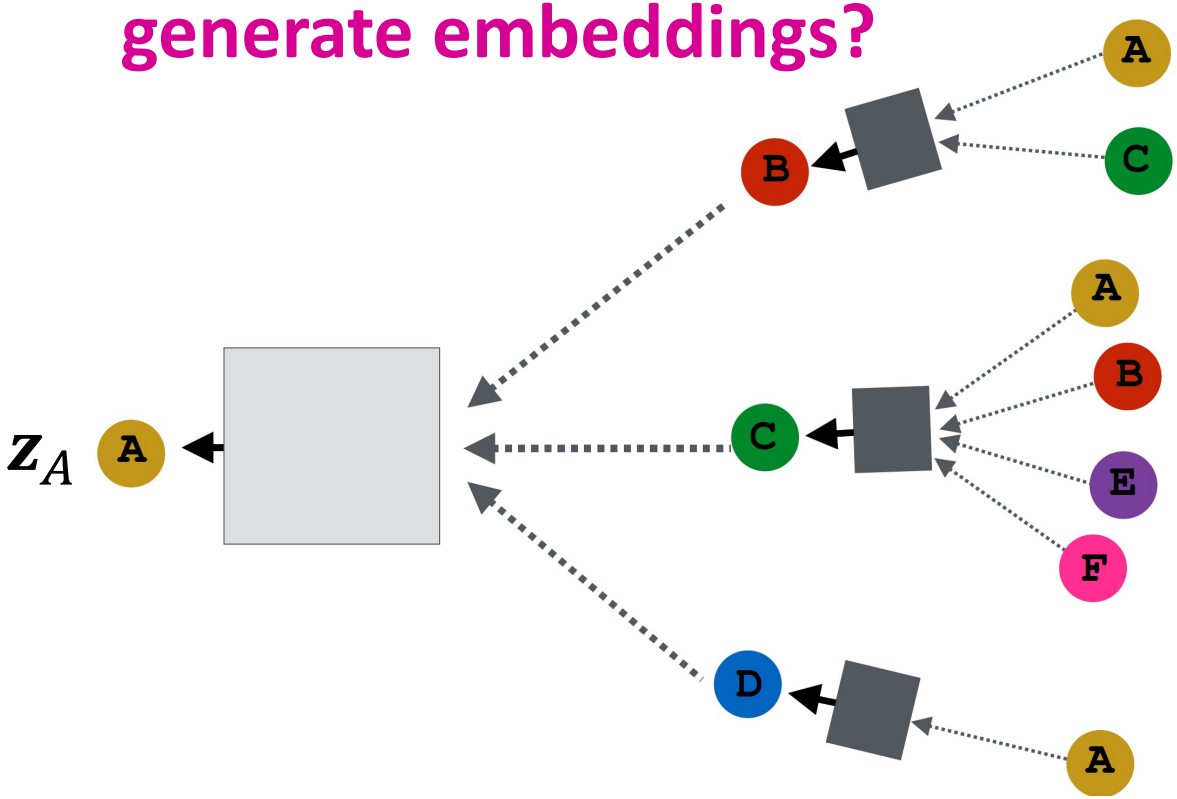


Adjacency matrix A_2



Training the Model

How do we train the GCN to generate embeddings?



Need to define a loss function on the embeddings.

Model Parameters

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

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

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

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

Final node embedding

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

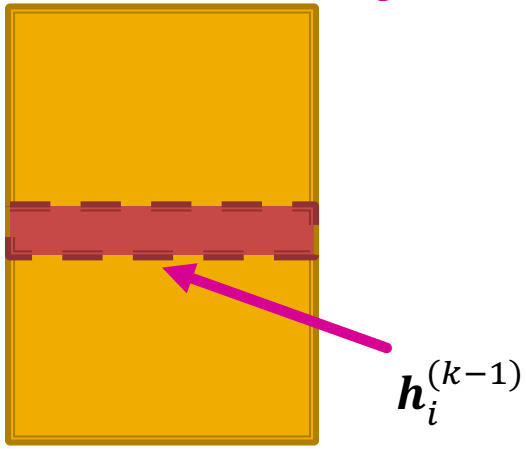
h_v^k: the hidden representation of node v at layer k

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

Matrix Formulation (1)

- **Many aggregations can be performed efficiently by (sparse) matrix operations**
- Let $H^{(k)} = [h_1^{(k)} \dots h_{|V|}^{(k)}]^{\mathrm{T}}$
- Then: $\sum_{u \in N_v} h_u^{(k)} = \mathrm{A}_{v,:}\mathrm{H}^{(k)}$
- Let D be diagonal matrix where $D_{v,v} = \mathrm{Deg}(v) = |N(v)|$
  - The inverse of D: D^{-1} is also diagonal: $D_{v,v}^{-1} = 1/|N(v)|$
- **Therefore,**

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



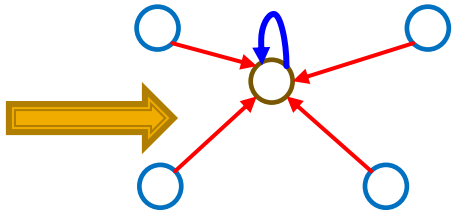
$$\sum_{u \in N(v)} \frac{h_u^{(k-1)}}{|N(v)|} \Longrightarrow H^{(k+1)} = D^{-1}AH^{(k)}$$

Matrix Formulation (2)

- Re-writing update function in matrix form:

$$H^{(k+1)} = \sigma(\tilde{A}H^{(k)}W_k^{\mathrm{T}} + H^{(k)}B_k^{\mathrm{T}})$$

where $\tilde{A} = D^{-1}A$



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

  - Red: neighborhood aggregation
  - Blue: self transformation

- In practice, this implies that efficient sparse matrix multiplication can be used ($\tilde{A}$ is sparse)

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

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):

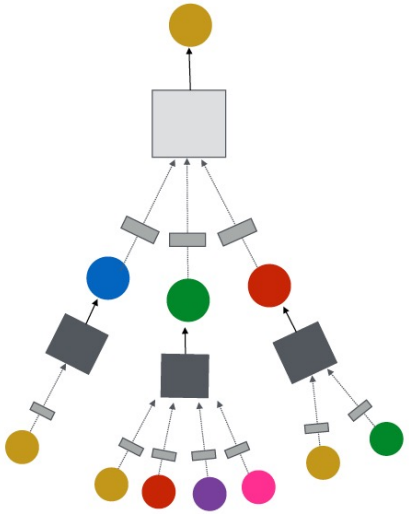
$$\min_{\Theta} \mathcal{L}(\boldsymbol{y}, f(\boldsymbol{z}_v))$$

  - $\boldsymbol{y}$: 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!**

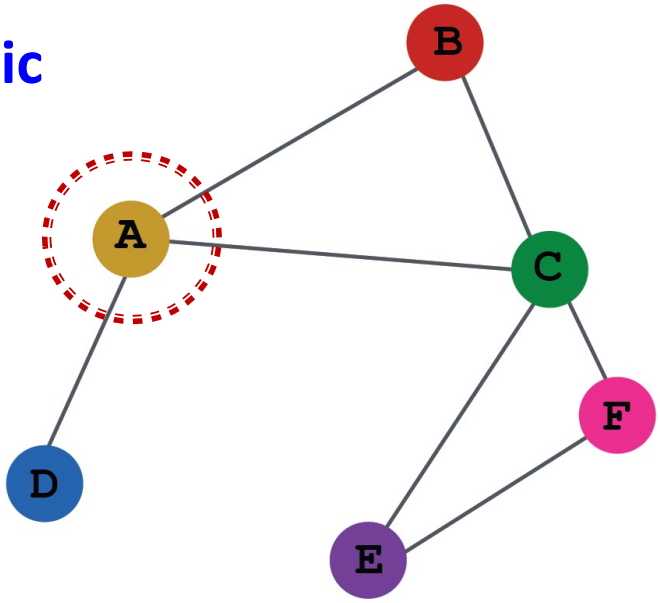
Supervised Training

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)

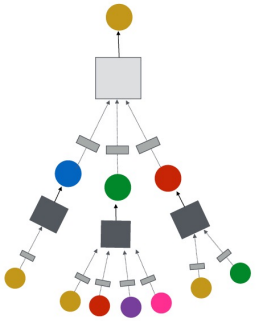
$$\mathcal{L} = \sum_{v \in V} y_v \log(\sigma(z_v^T \theta)) + (1 - y_v) \log(1 - \sigma(z_v^T \theta))$$

Encoder output: node embedding

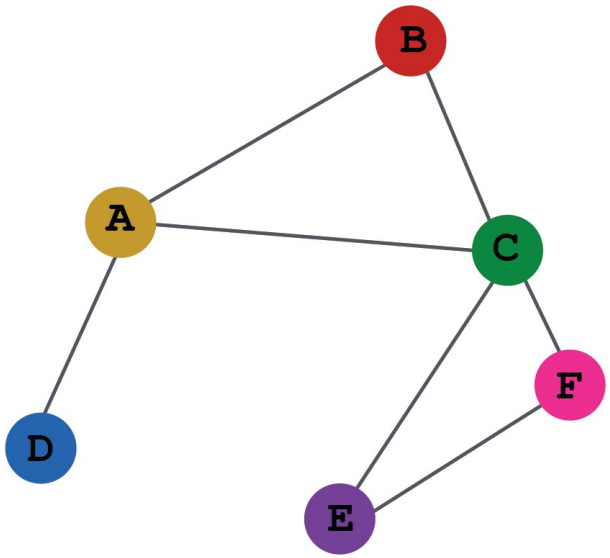
Classification weights

Node class label

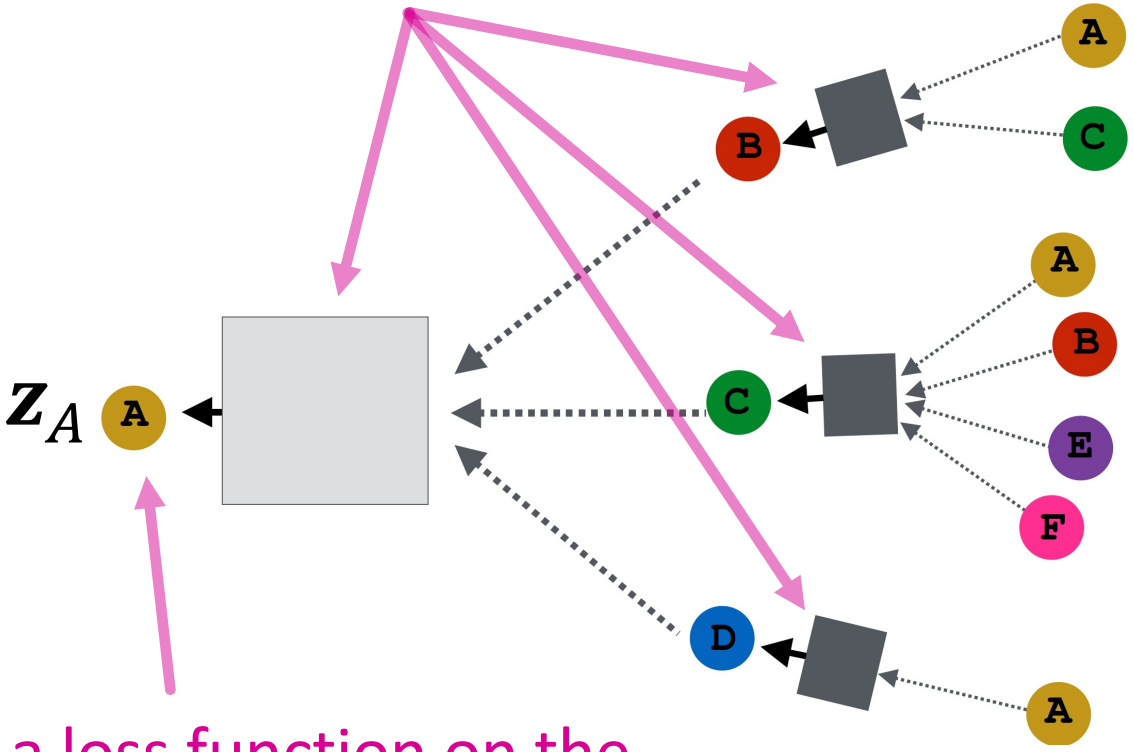
Safe or toxic drug?



Model Design: Overview

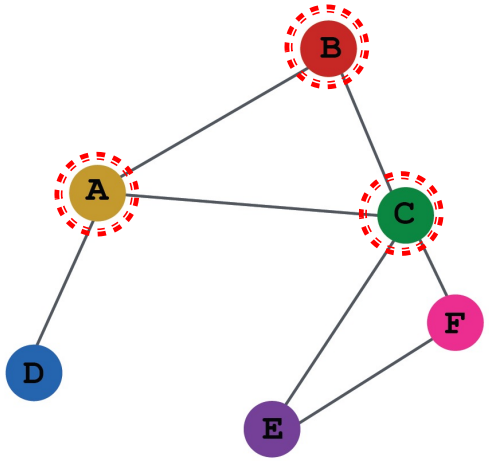


(1) Define a neighborhood aggregation function



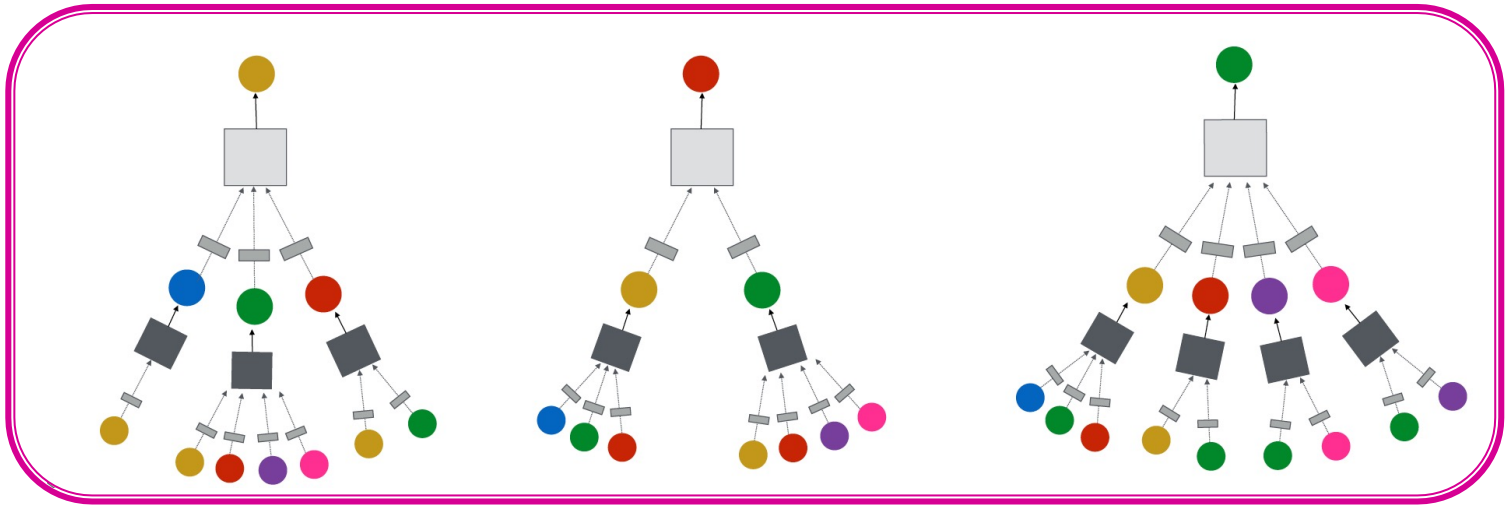
(2) Define a loss function on the embeddings

Model Design: Overview

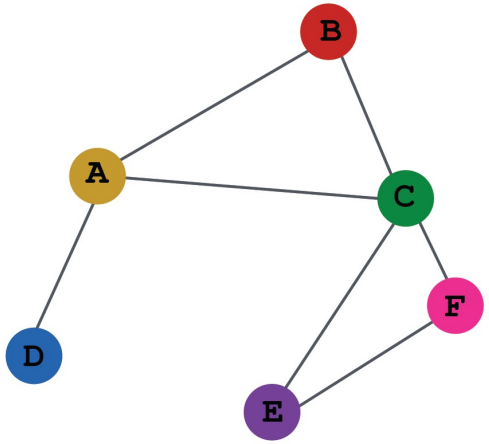


INPUT GRAPH

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



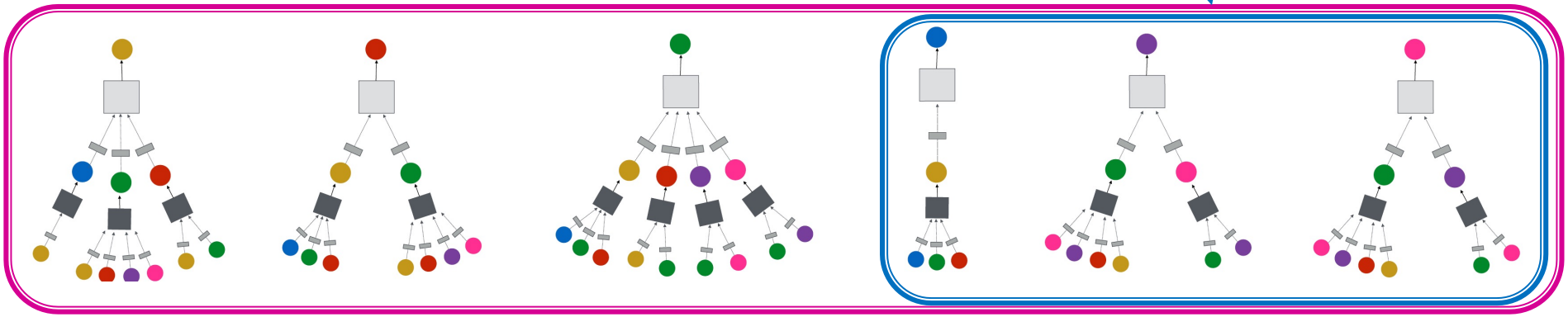
Model Design: Overview



INPUT GRAPH

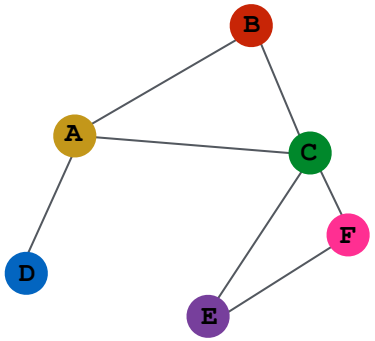
(4) Generate embeddings for nodes as needed

Even for nodes we never trained on!

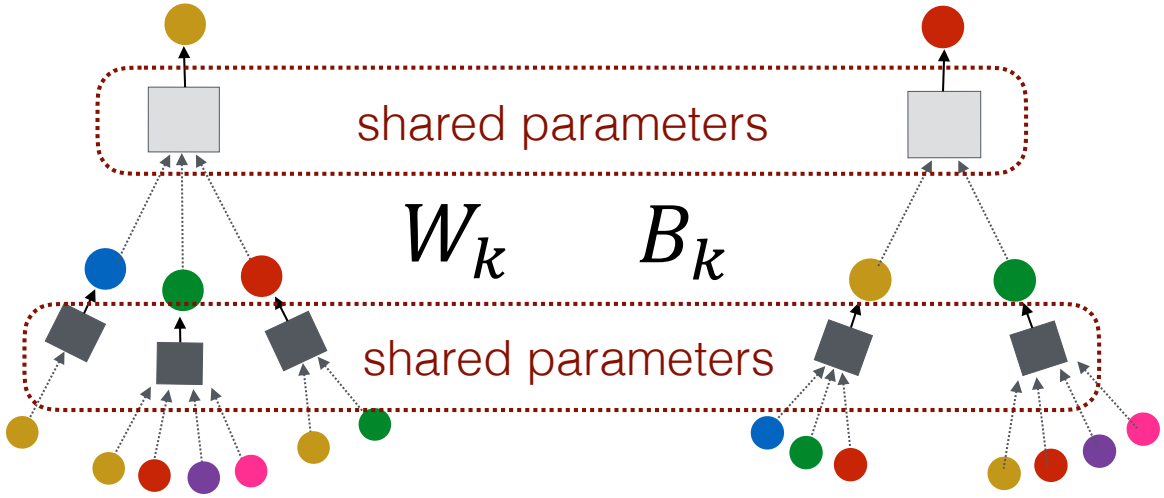


Inductive Capability

- **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**!



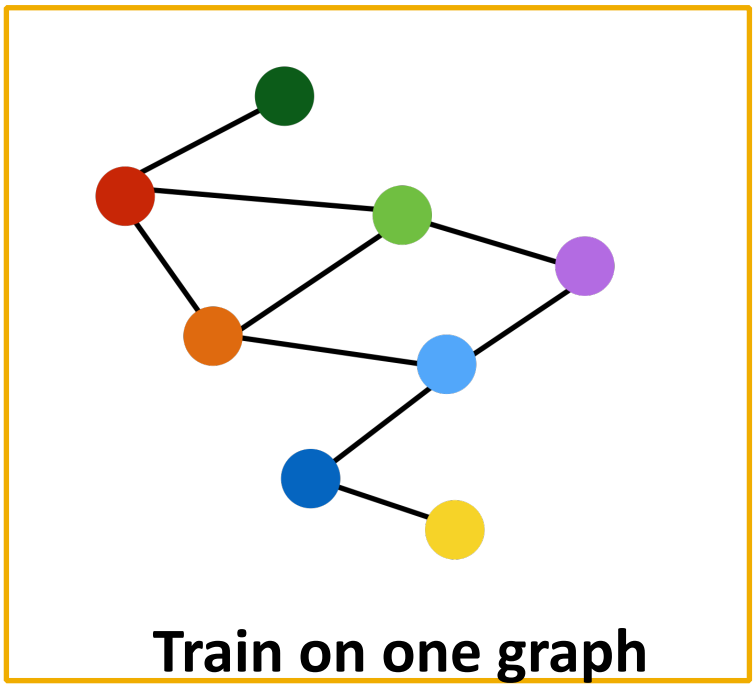
INPUT GRAPH



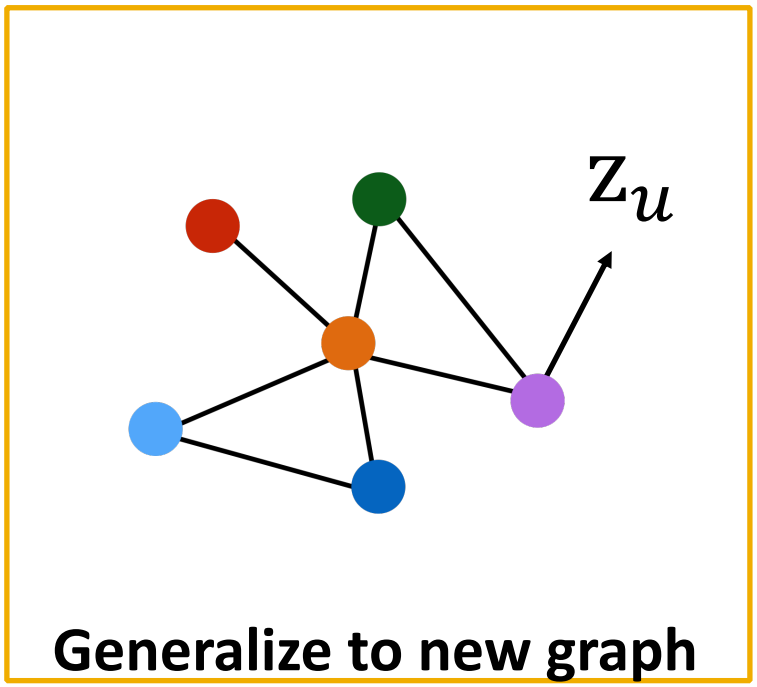
Compute graph for node A

Compute graph for node B

Inductive Capability: New Graphs



Train on one graph

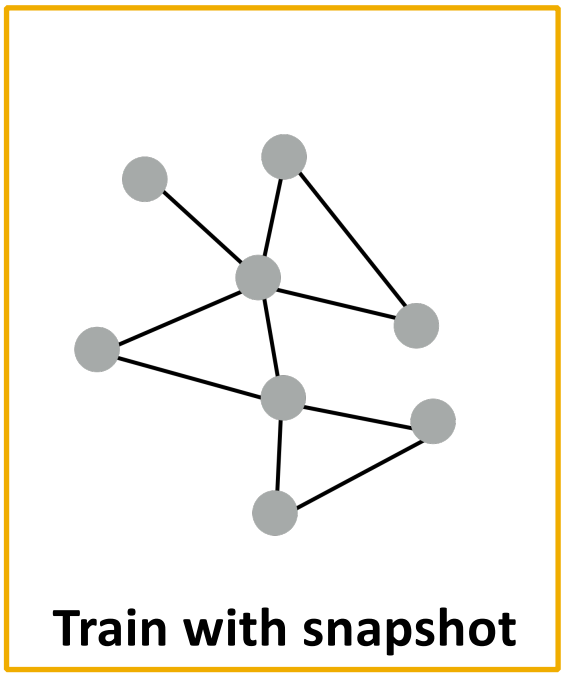


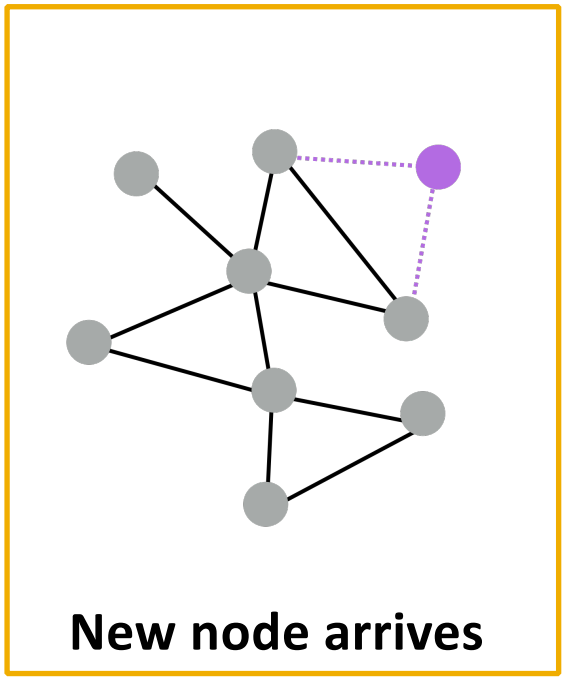
Generalize to new graph

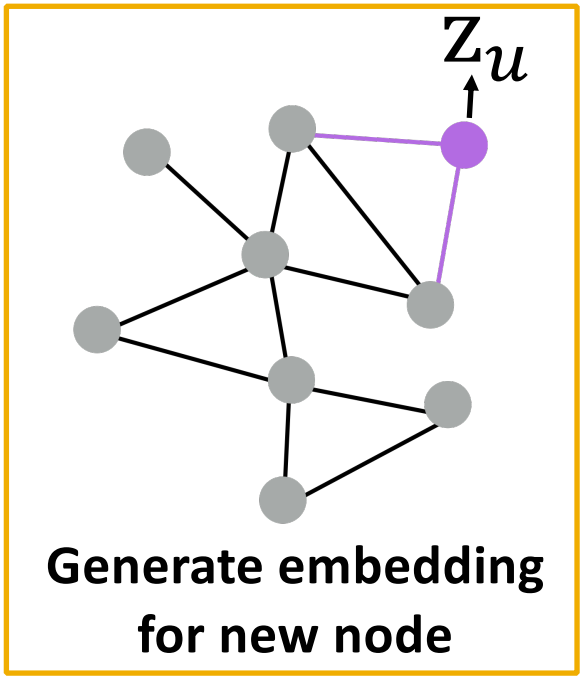
Inductive node embedding → Generalize to entirely unseen graphs

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

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"