LECTURE 14: GRAPH NEURAL NETWORKS

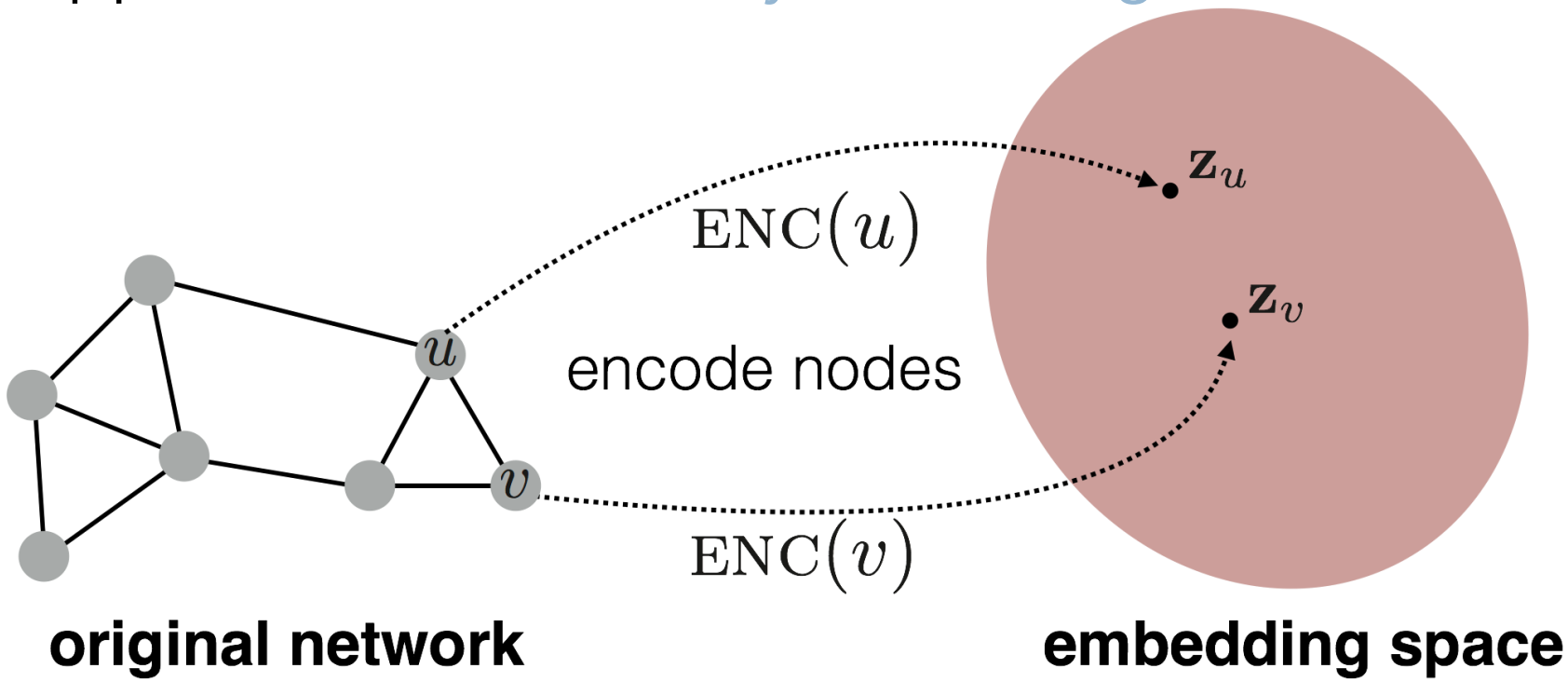
Prof. Pan Hui

CSIT 6000K: Social Networks and Social Computing: A Data Science Perspective

Thursdays 07:30 PM - 10:20 PM

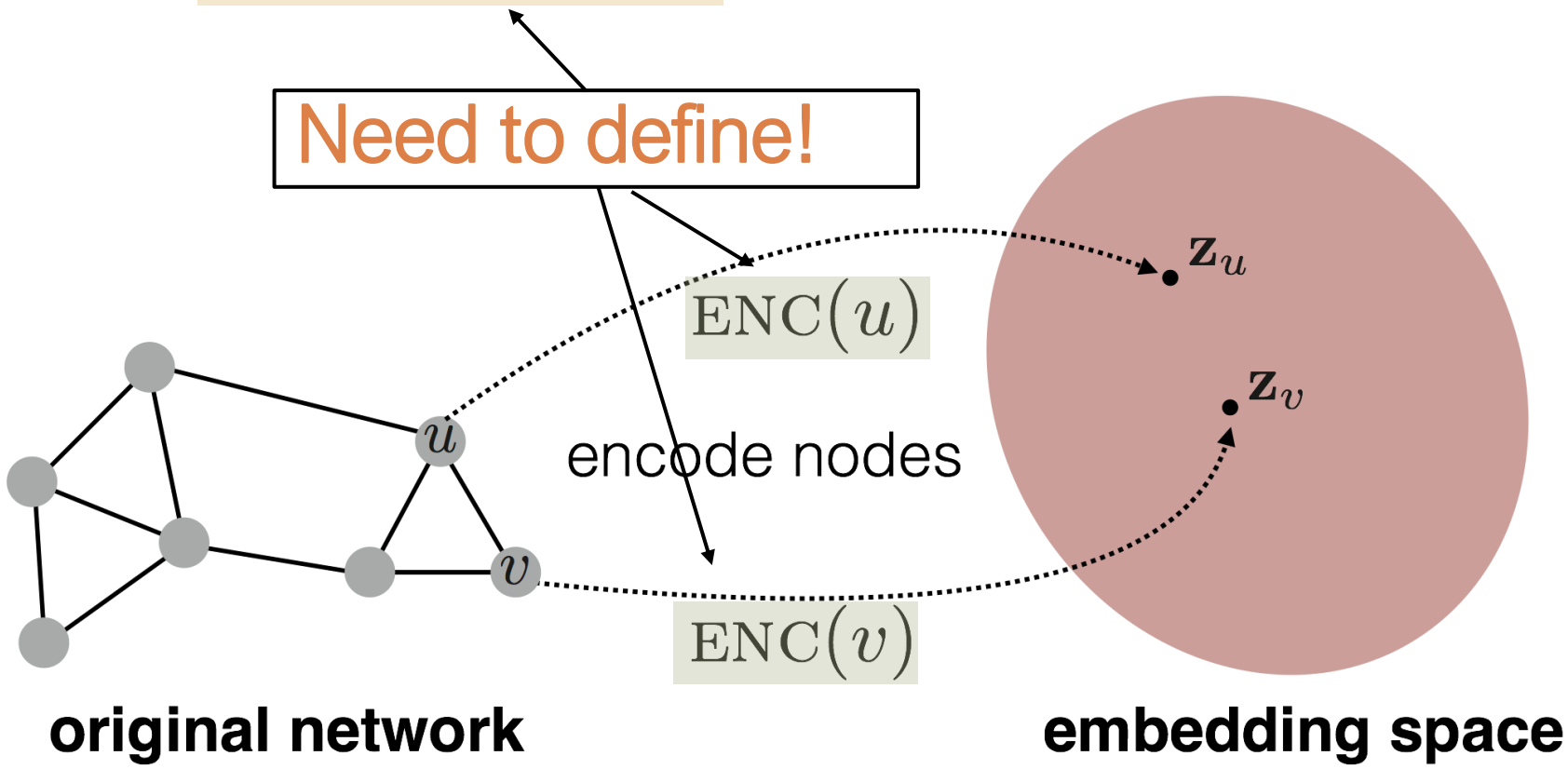
Embedding Nodes

- 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.

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

d-dimensional embedding

node in the input graph

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

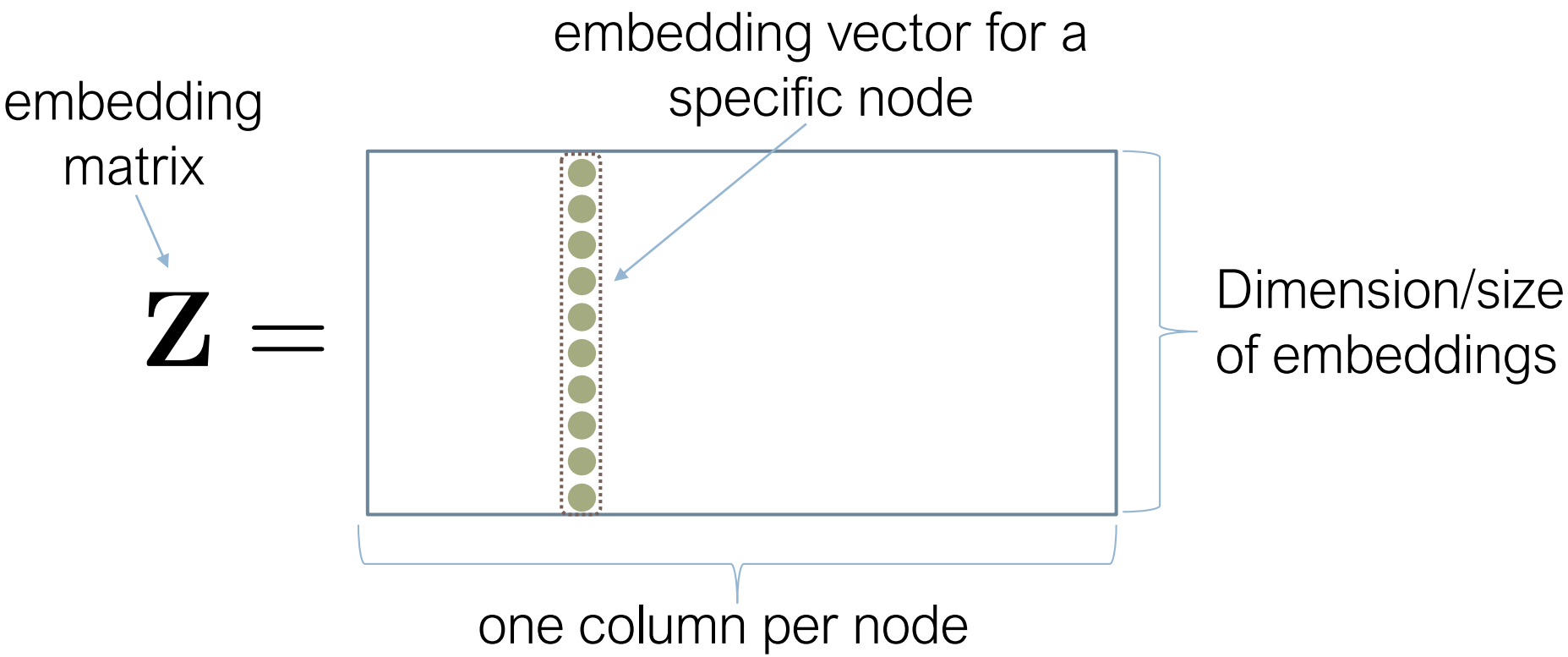
$$\mathrm{similarity}(u, v) \approx \mathbf{z}_v^\top \mathbf{z}_u$$

Similarity of u and v in the original network

dot product between node embeddings

From "Shallow" to "Deep"

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



From "Shallow" to "Deep"

- Limitations of shallow encoding:
  - **O(|V|) parameters are needed**: there no parameter sharing and every node has its own unique embedding vector.
  - **Inherently "transductive":** It is impossible to generate embeddings for nodes that were 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.

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

The Basics: Graph Neural Networks

Based on material from:

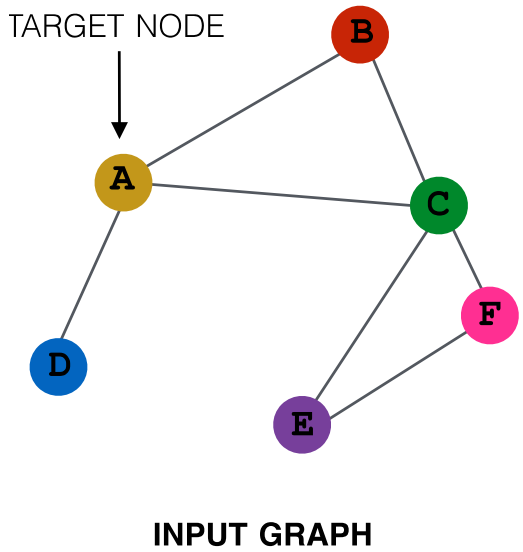
- Hamilton et al. 2017. Representation Learning on Graphs: Methods and Applications. *IEEE Data Engineering Bulletin on Graph Systems*.
- Scarselli et al. 2005. The Graph Neural Network Model. *IEEE Transactions on Neural Networks*.

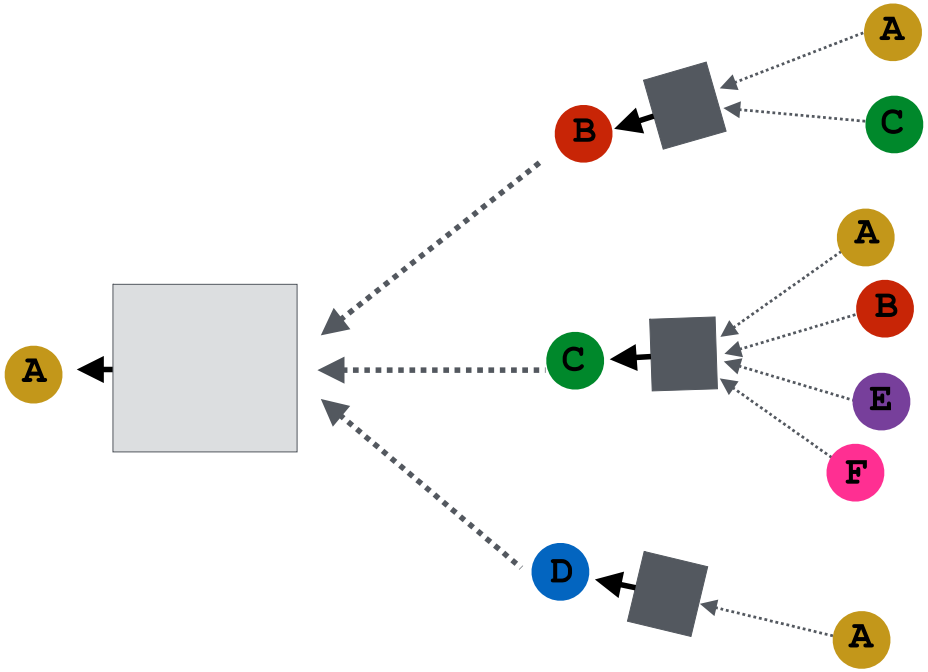
Setup

- Assume we have a graph G:
  - V is the vertex set.
  - $\mathbf{A}$ is the adjacency matrix (assume binary).
  - $\mathbf{X} \in \mathrm{R}^{m \times |V|}$ **is a matrix of node features.**
    - Categorical attributes, text, image data
      - E.g., profile information in a social network.
    - Node degrees, clustering coefficients, etc.
    - Indicator vectors (i.e., one-hot encoding of each node)

Neighborhood Aggregation

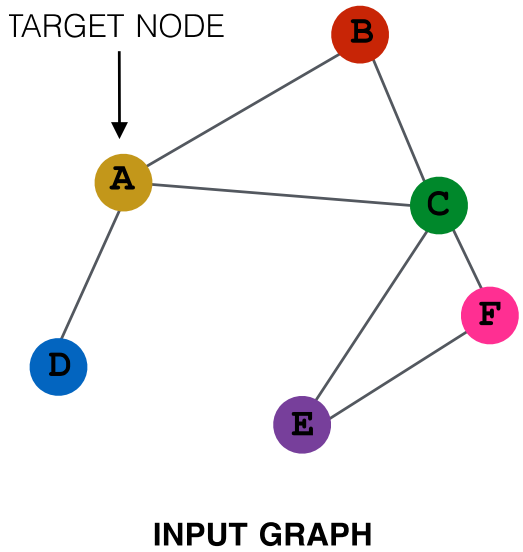
- **Key idea:** Generate node embeddings based on local neighborhoods.

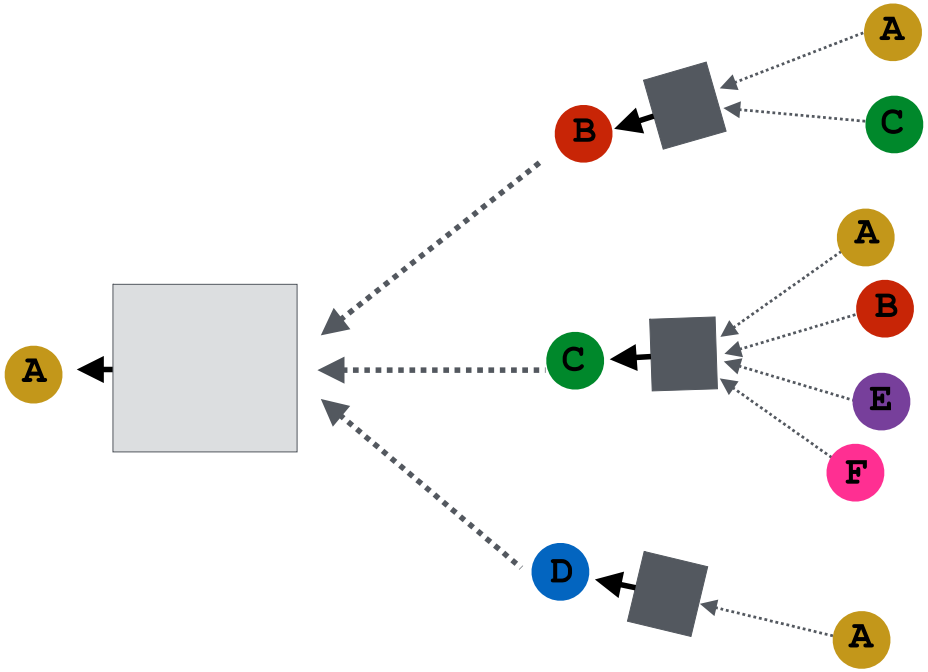




Neighborhood Aggregation

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

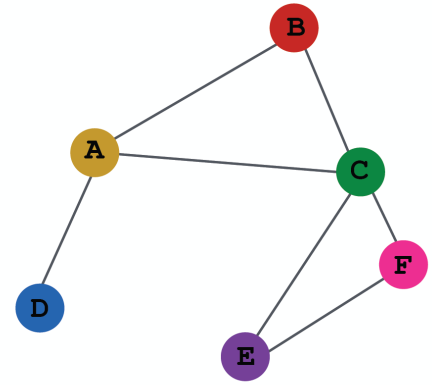




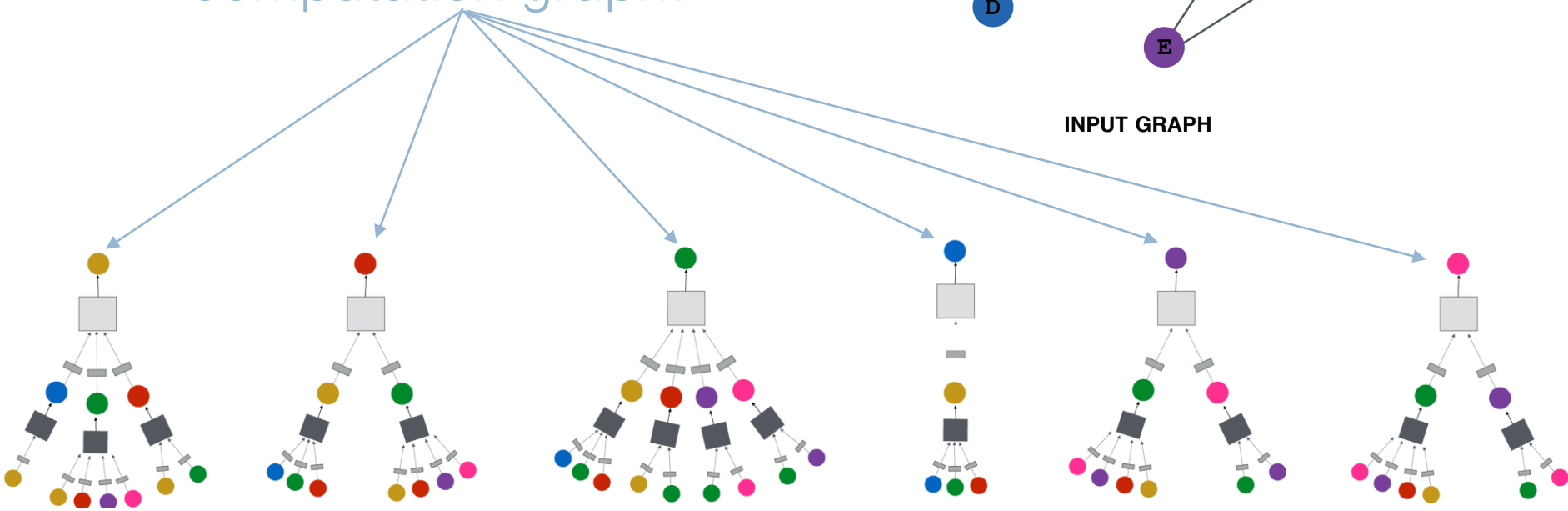
Neighborhood Aggregation

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

Every node defines a unique computation graph!

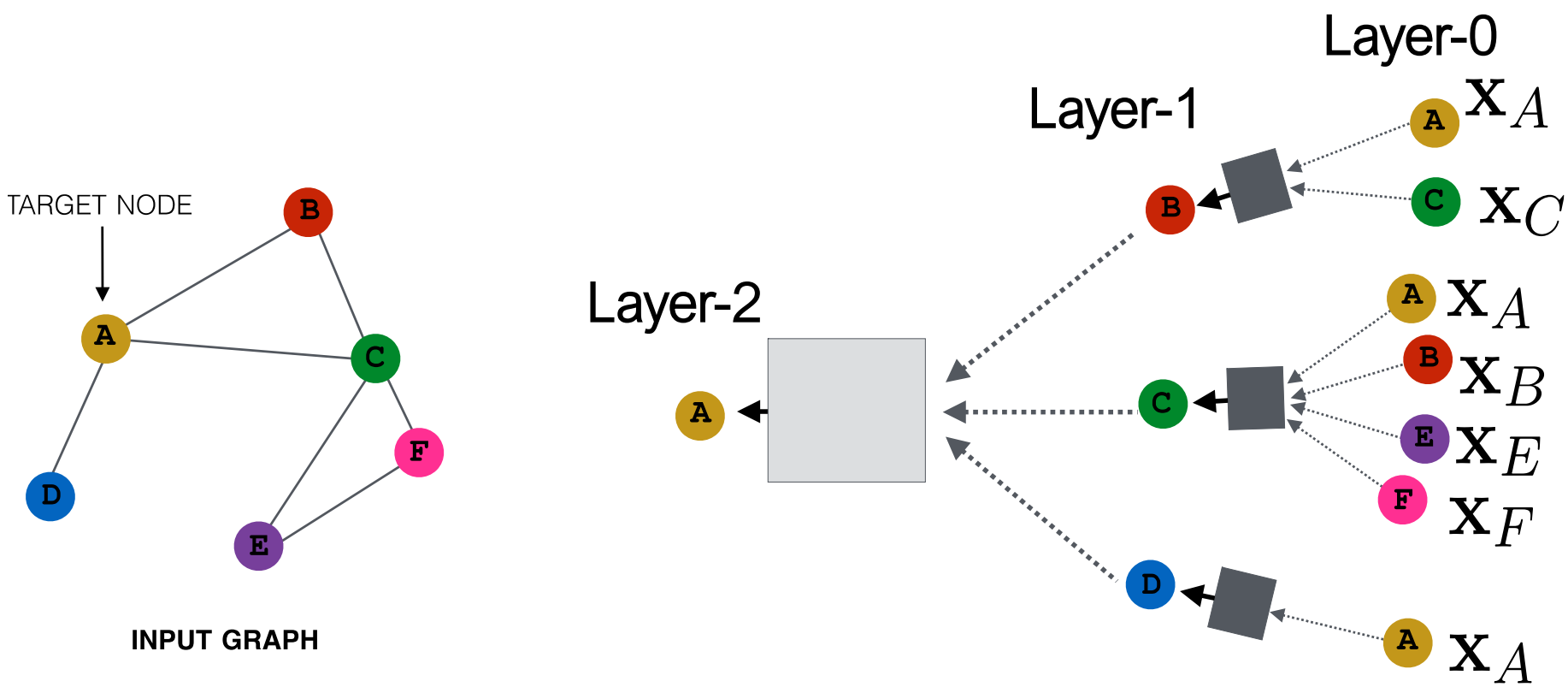


INPUT GRAPH



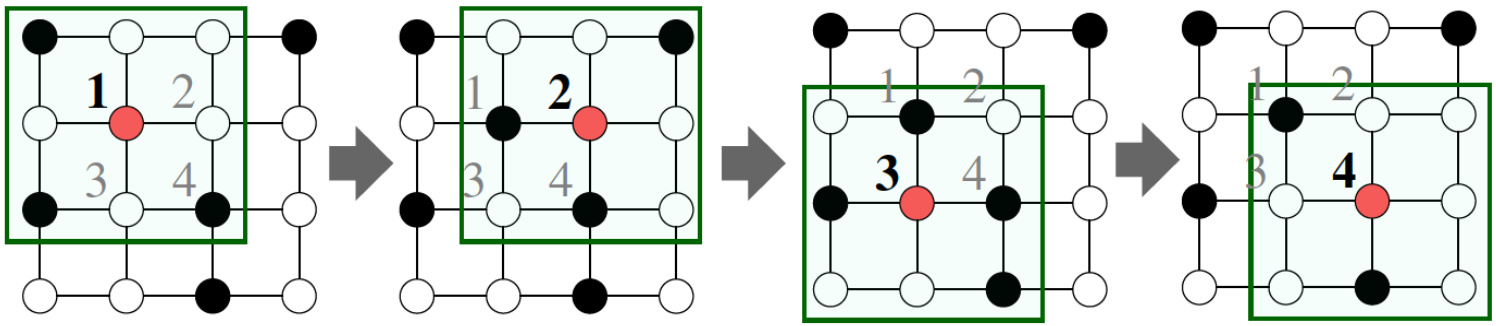
Neighborhood Aggregation

- 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.



Neighborhood "Convolutions"

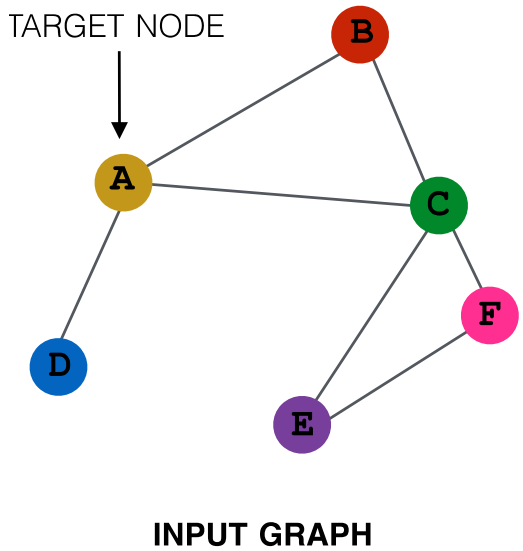
- Neighborhood aggregation can be viewed as a center-surround filter.



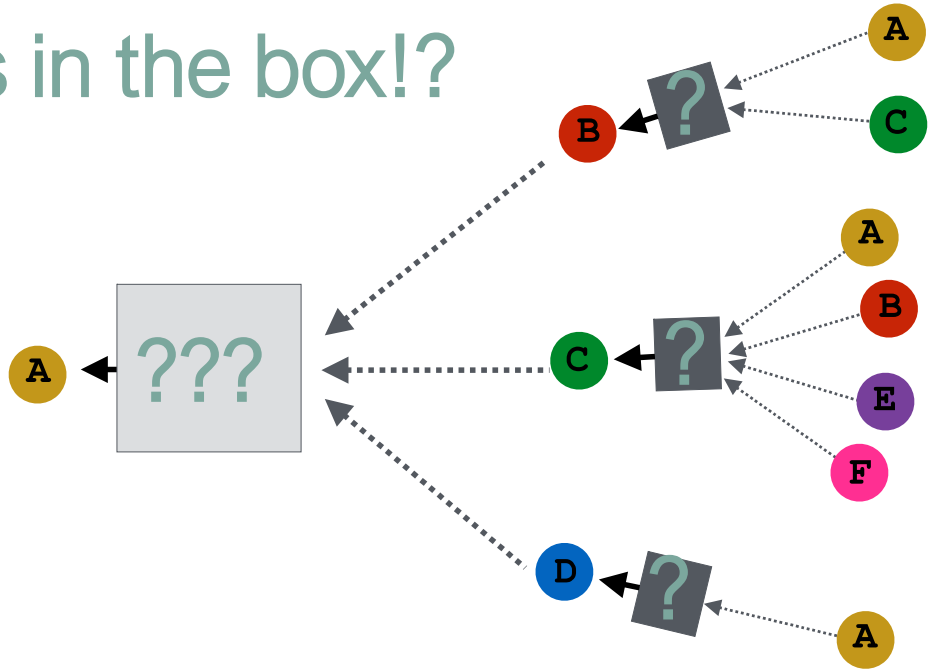
- Mathematically related to spectral graph convolutions (see Bronstein et al., 2017)

Neighborhood Aggregation

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

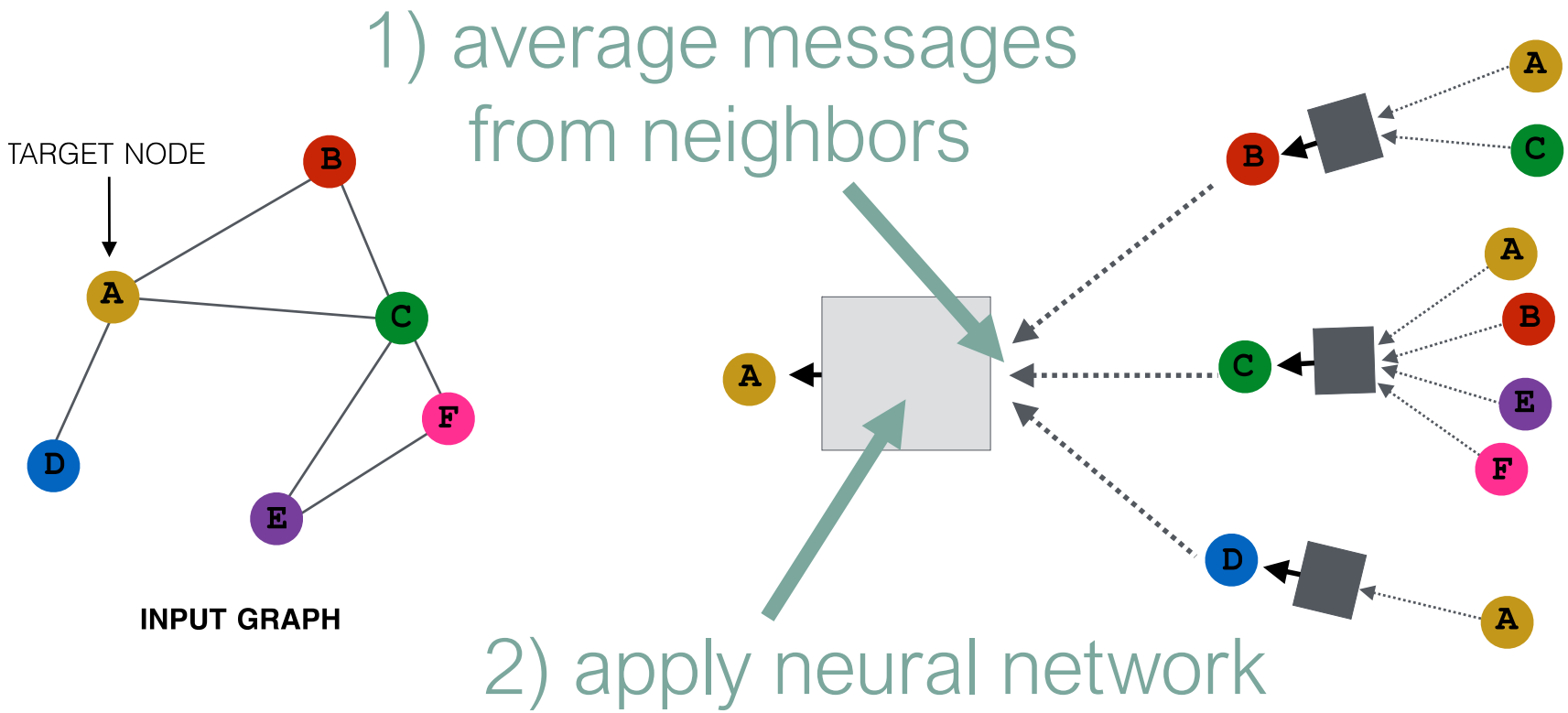


what's in the box!?



Neighborhood Aggregation

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



The Math: Deep Encoder

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

Initial "layer 0" embeddings are equal to node features

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

previous layer embedding of v

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

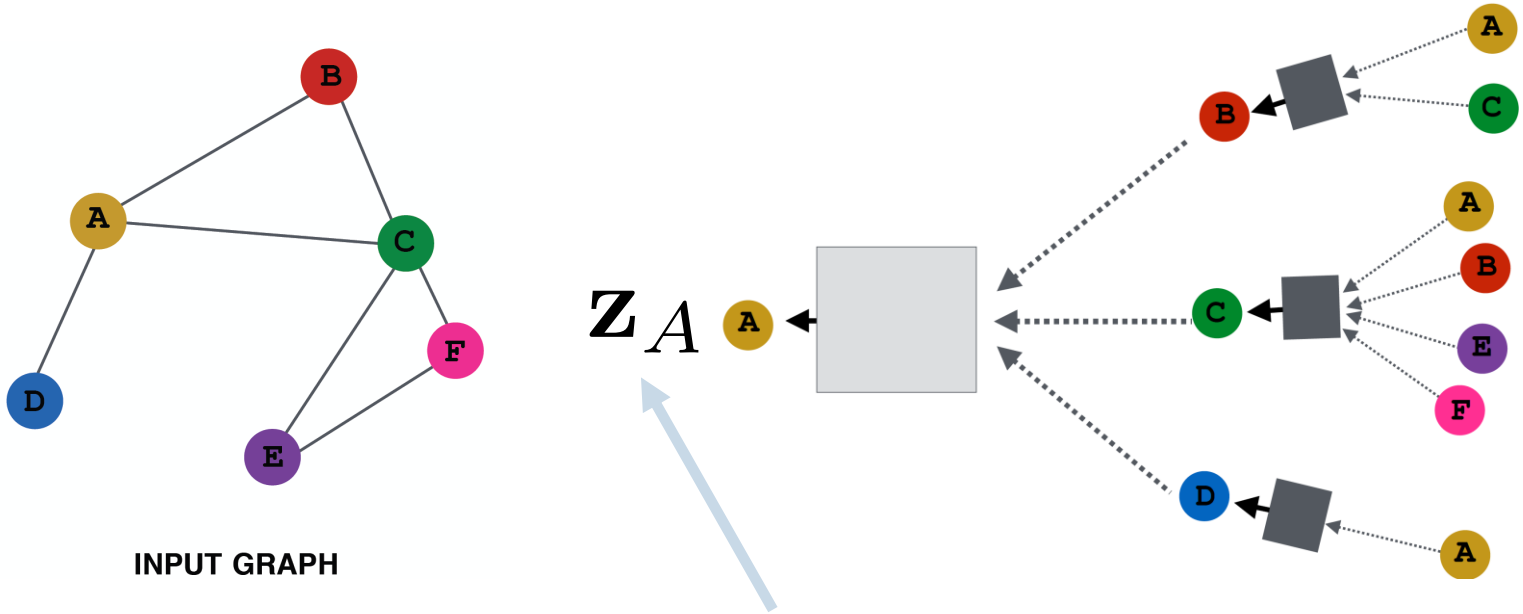
kth layer embedding of v

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

average of neighbor's previous layer embeddings

Training the Model

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



Need to define a loss function on the embeddings, $\mathbf{L}(\mathbf{z}_u)$!

Training the Model

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

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

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

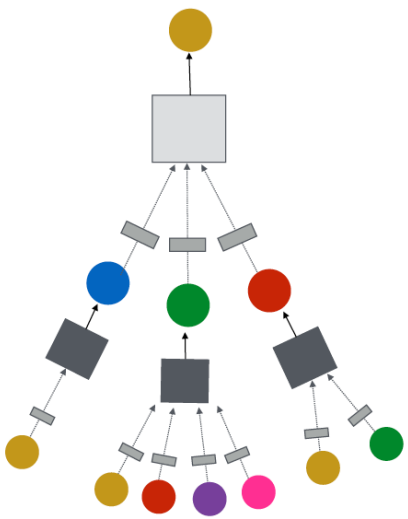
$$\boxed{\mathbf{z}_v = \mathbf{h}_v^K}$$

- 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.

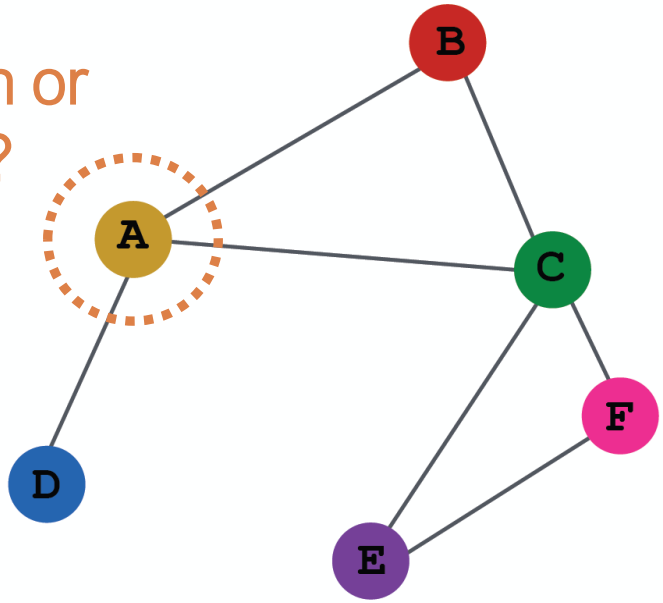
Training the Model

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

Human or bot?



Human or bot?



e.g., an online social network

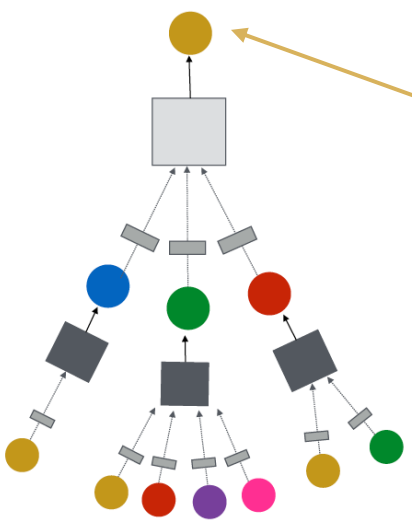
Training the Model

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

classification weights

Human or bot?

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

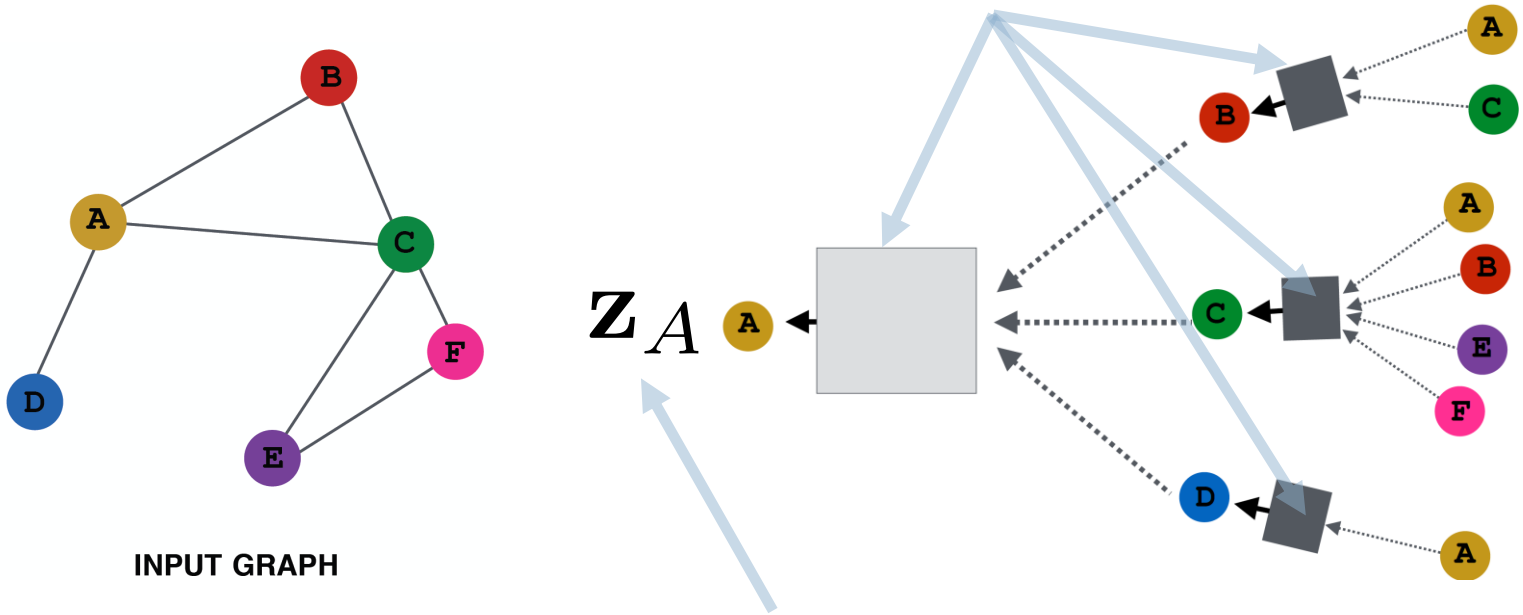


output node embedding

node class label

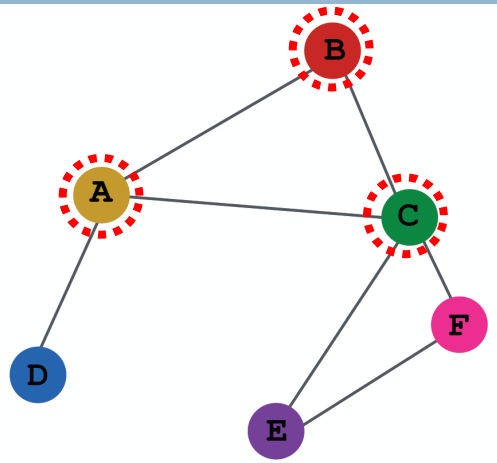
Overview of Model Design

1) Define a neighborhood aggregation function.



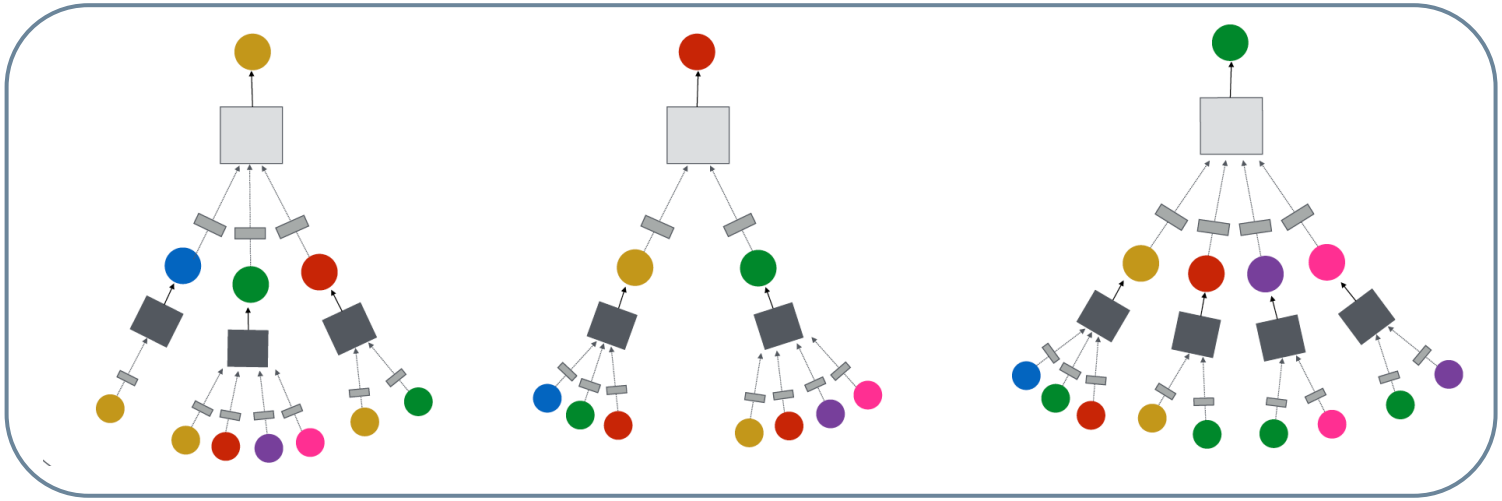
2) Define a loss function on the embeddings, $L(z_u)$

Overview of Model Design

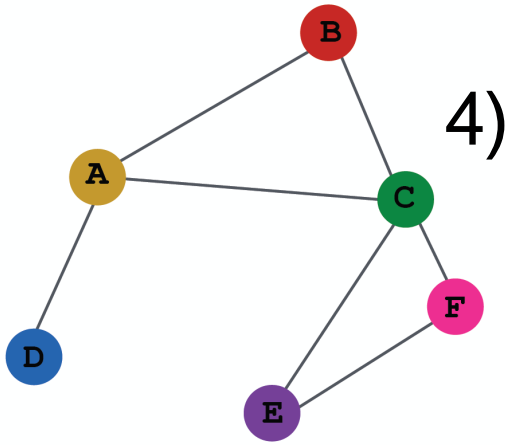


INPUT GRAPH

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



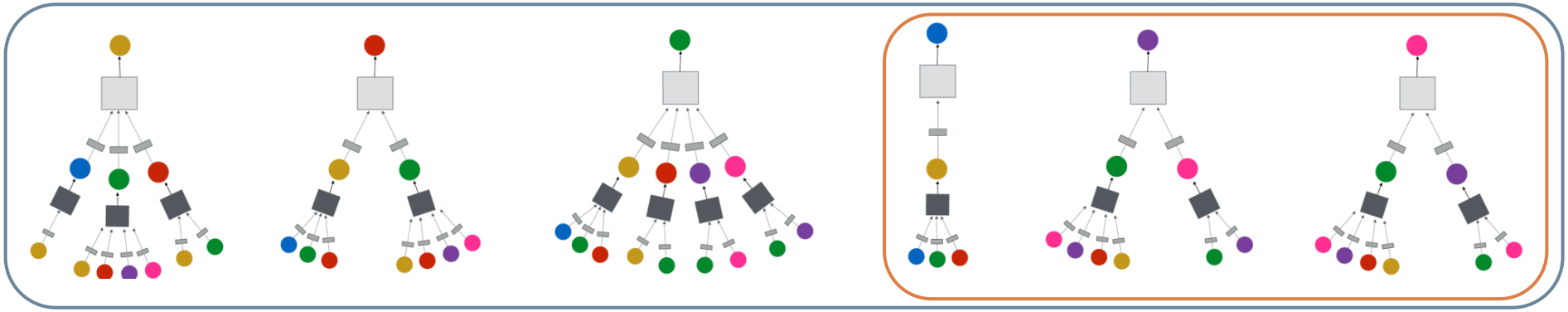
Overview of Model



INPUT GRAPH

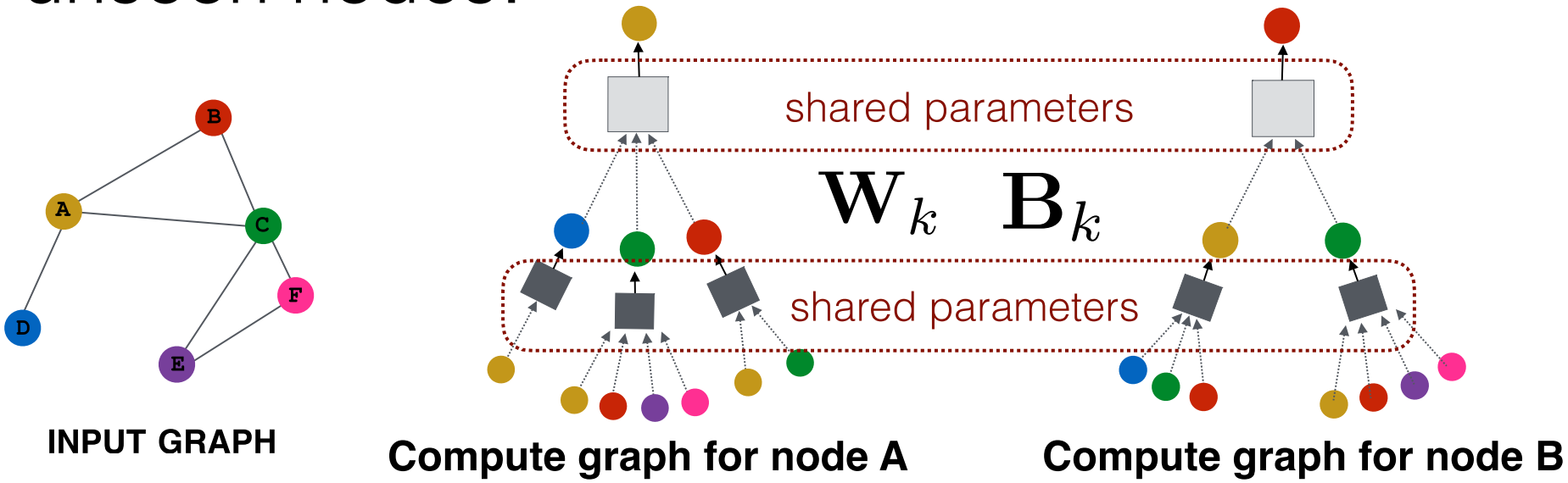
4) Generate embeddings for nodes as needed

Even for nodes we never trained on!!!!

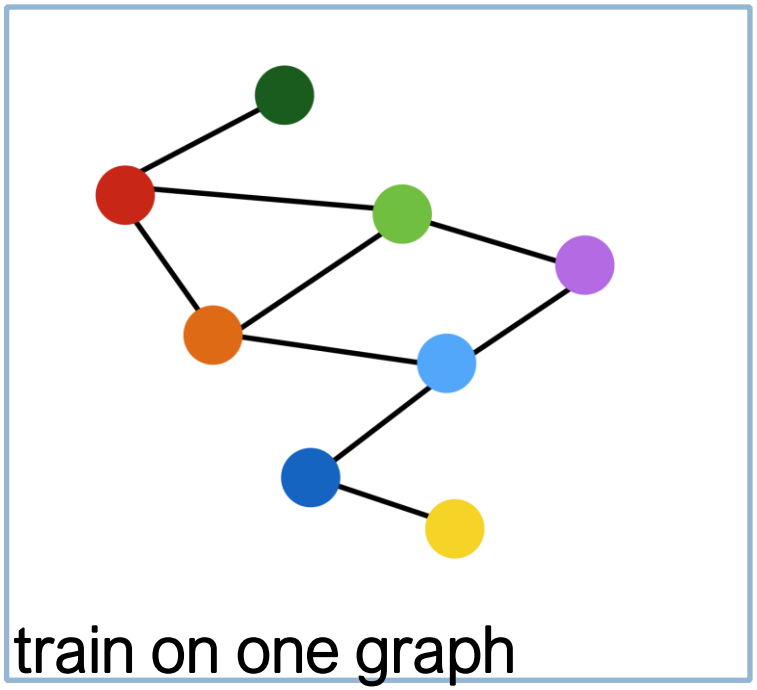


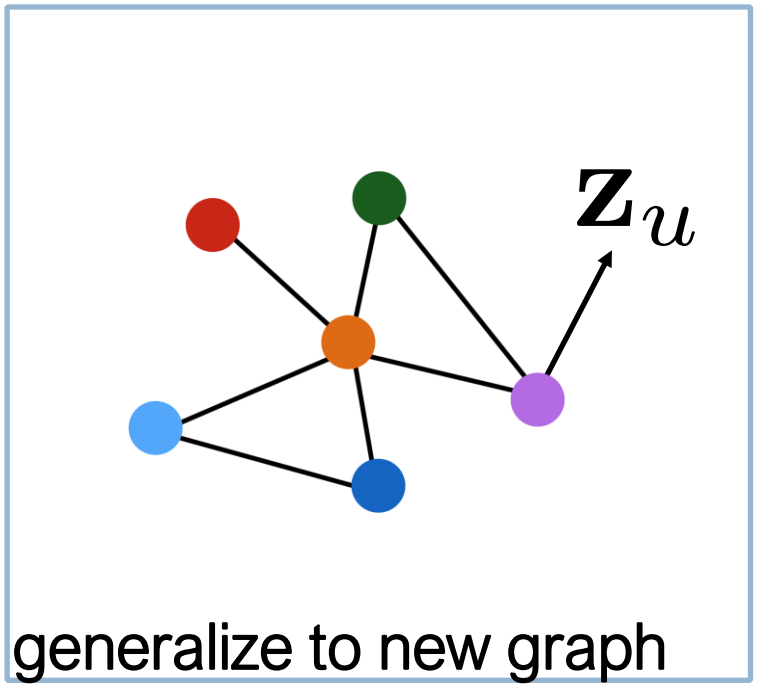
Inductive Capability

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



Inductive Capability: New Graphs

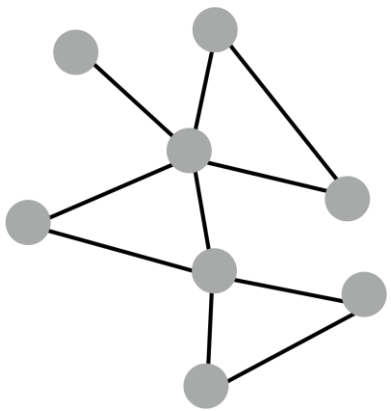




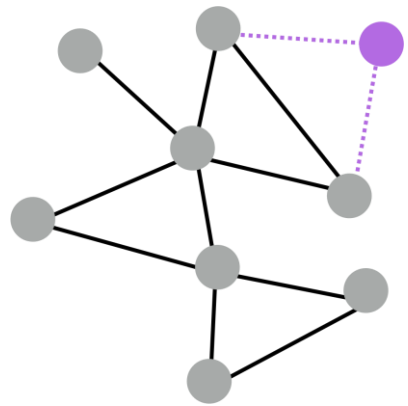
Inductive node embedding → generalize to entirely unseen graphs

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

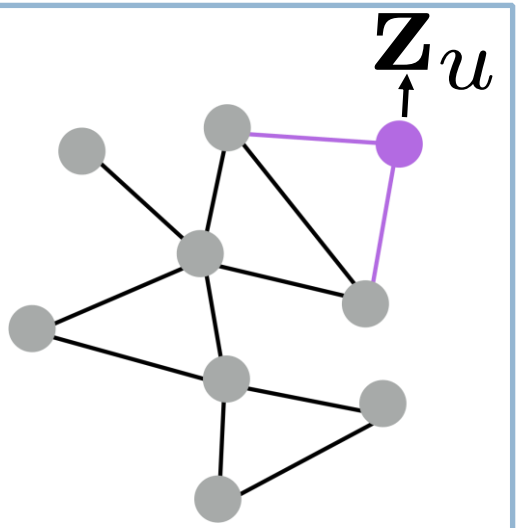
Inductive Capability: New Nodes



train with snapshot



new node arrives



generate embedding for new node

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.

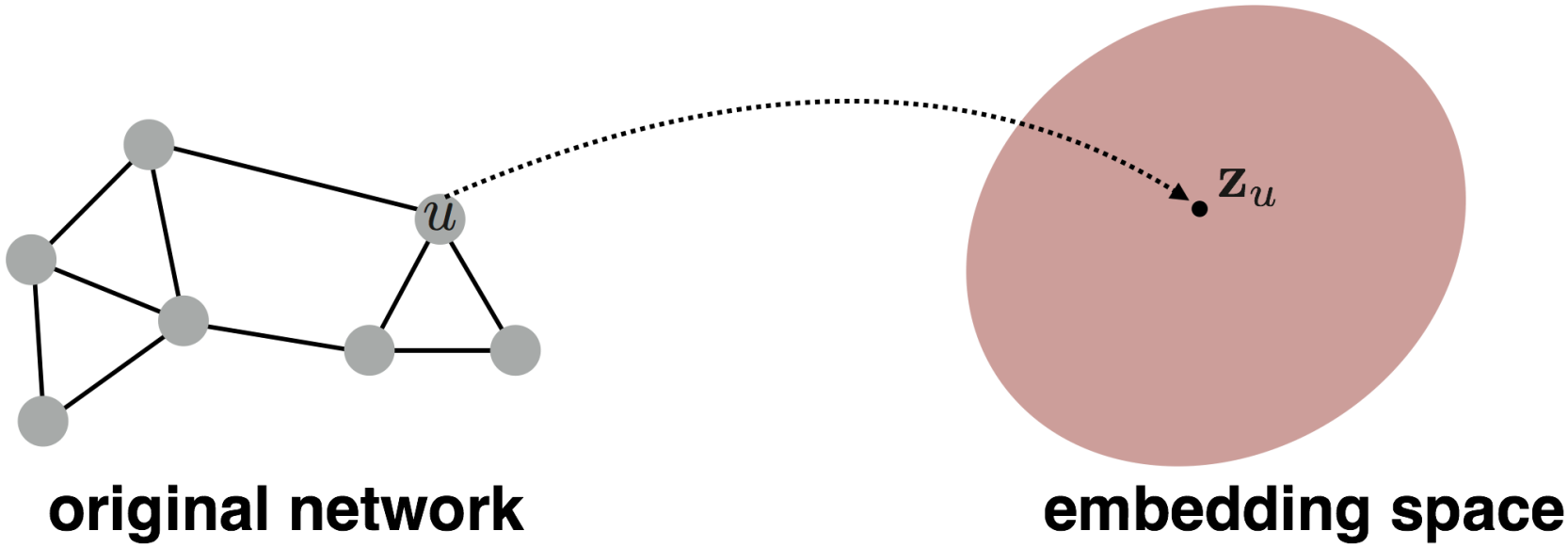
Subgraph Embeddings

Based on material from:

- Duvenaud et al. 2016. Convolutional Networks on Graphs for Learning Molecular Fingerprints. *ICML.*
- Li et al. 2016. Gated Graph Sequence Neural Networks. *ICLR.*

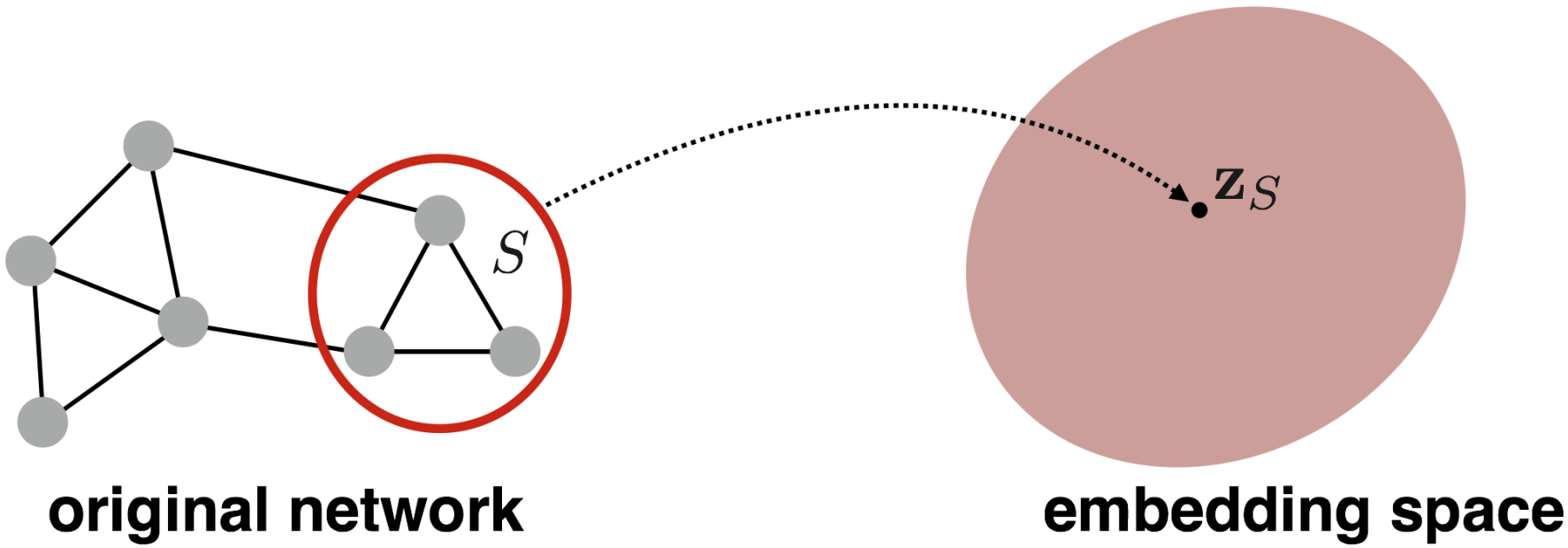
(Sub)graph Embeddings

- **So far we have focused on node-level embeddings…**



(Sub)graph Embeddings

- **But what about subgraph embeddings?**



Approach 1

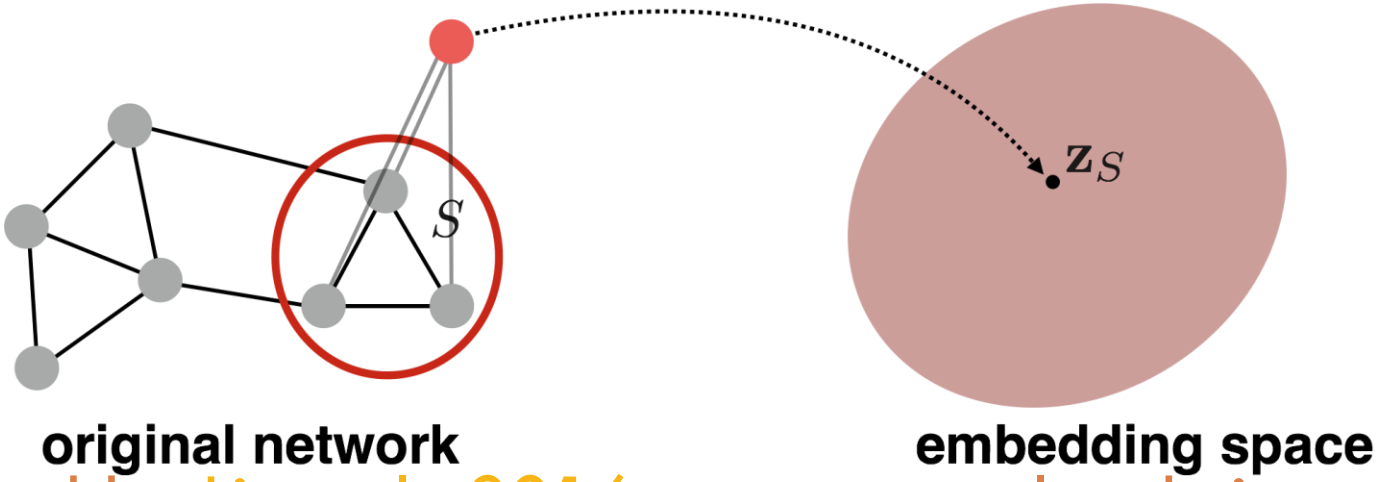
- **Simple idea:** Just sum (or average) the node embeddings in the (sub)graph

$$\mathbf{z}_S = \sum_{v \in S} \mathbf{z}_v$$

- Used by Duvenaud et al., 2016 to classify molecules based on their graph structure.

Approach 2

- **Idea:** Introduce a **"virtual node"** to represent the subgraph and run a standard graph neural network.



- Proposed by Li et al., 2016 as a general technique for subgraph embedding.