Machine Learning with Graphs (MLG)

Graph Neural Networks (GNN)

The basic form of neighborhood aggregation

Cheng-Te Li (李政德)

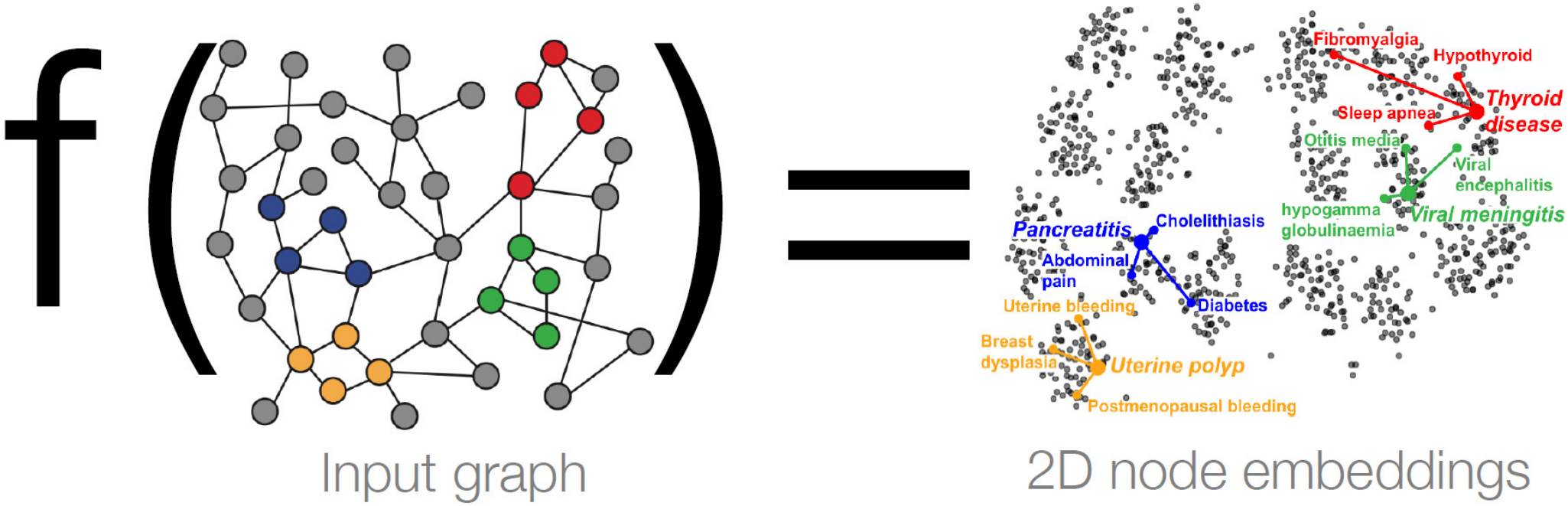
Institute of Data Science

National Cheng Kung University

chengte@mail.ncku.edu.tw

Node Embeddings

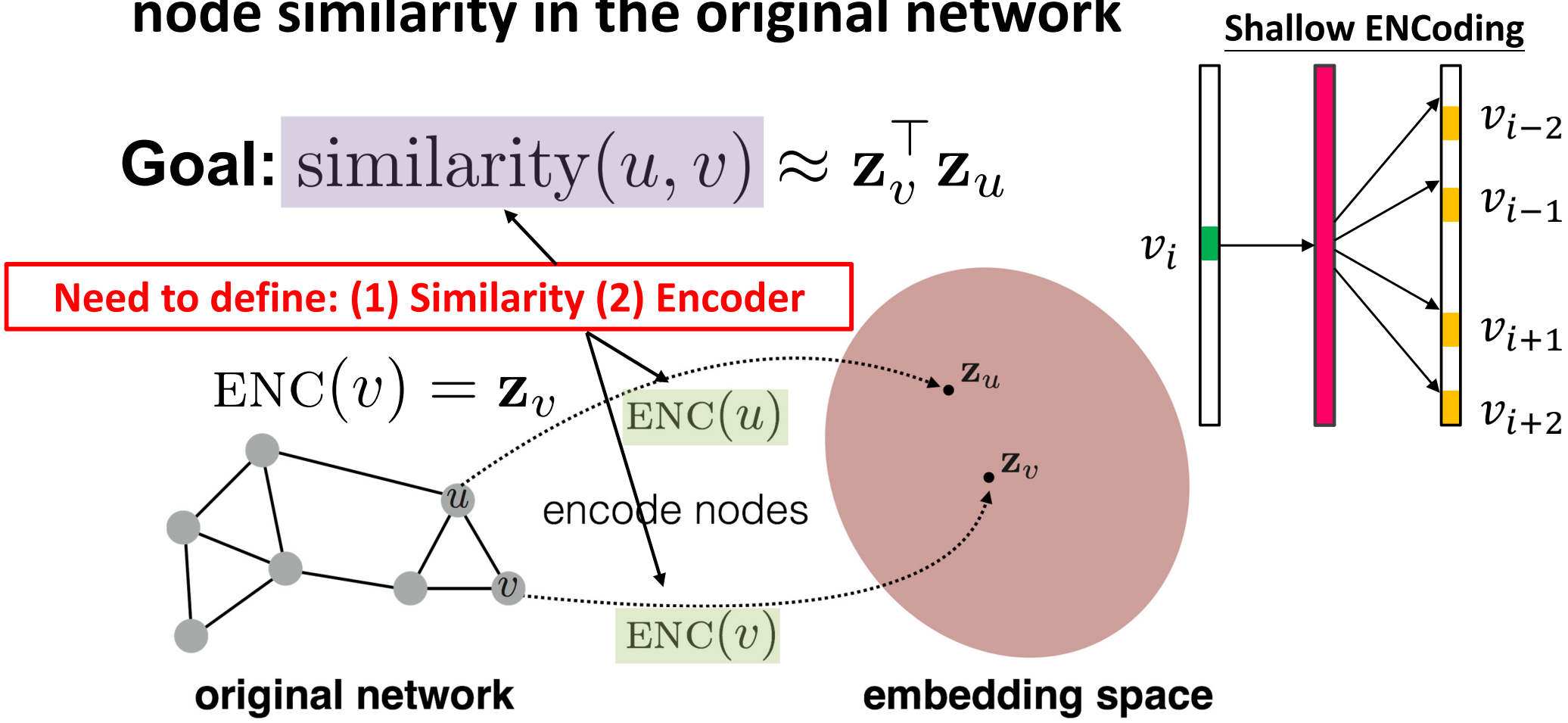
- **Intuition**: Map nodes to d-dimensional embeddings such that similar nodes in the graph are embedded close together



How to learn mapping function f?

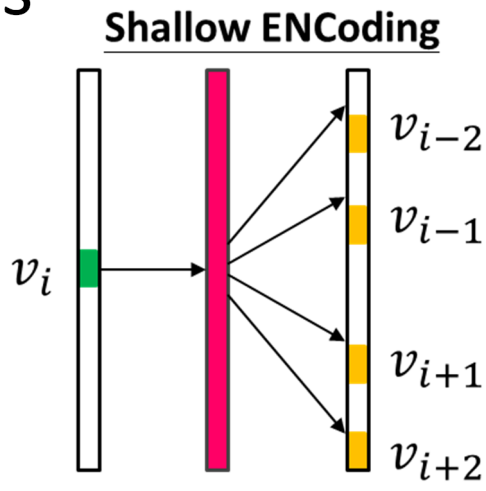
Embedding Nodes

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



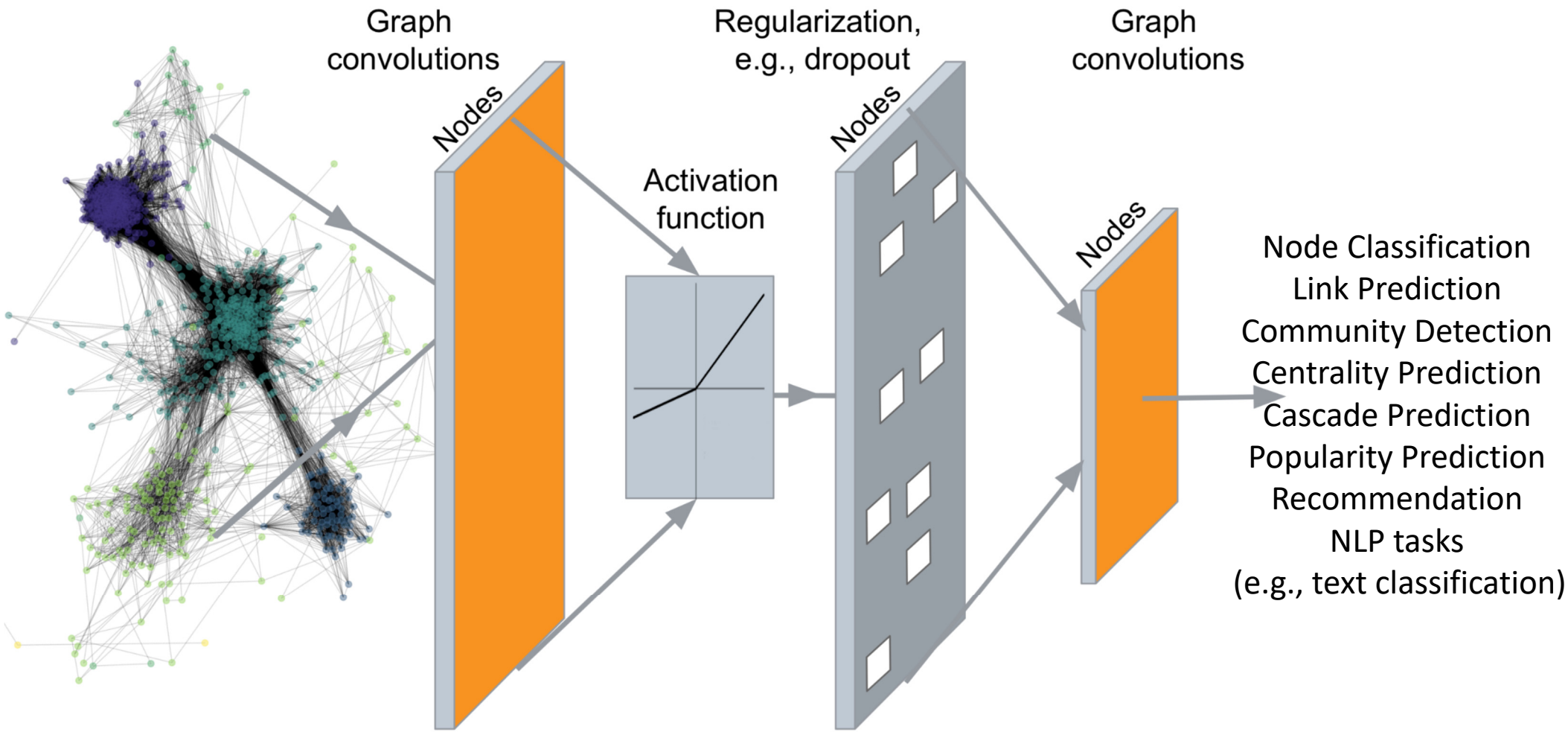
Limitations of Shallow Encoding

1) Require $|V|$ learnable parameters
   - No sharing of parameters between nodes
2) Inherently "transductive"
   - Cannot generate embeddings for nodes that are not seen during training
3) Unable to incorporate node features
   - Many graphs have node attributes that we can and should leverage



Graph Neural Networks (GNN)

Encoding every node in a graph through multiple layers of non-linear transformations of graph structure



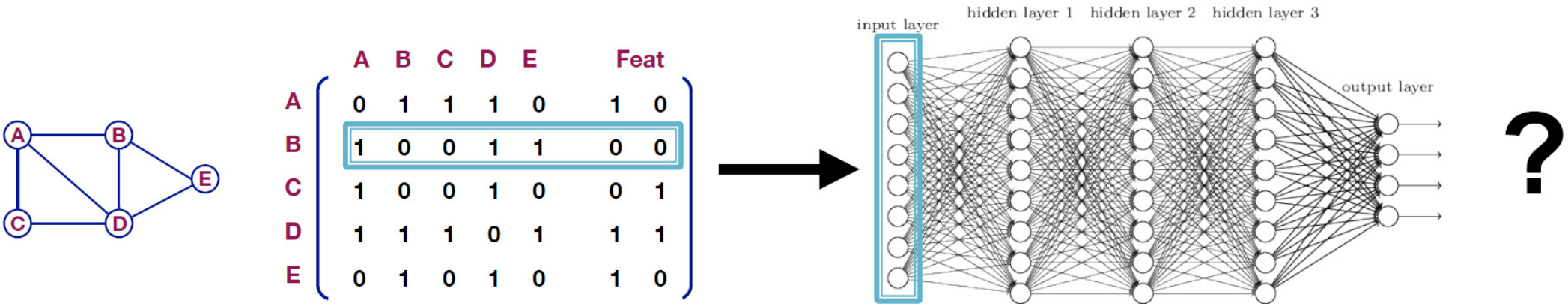
Notations

Assume we have a graph $G = (V, E)$

- V is the node set
- $\boldsymbol{A}$ is the adjacency matrix (assume binary)
- $\boldsymbol{X} \in \mathbb{R}^{m \times |V|}$ is a matrix of node features
- $v \in V$: a node in V
- $N(v)$: the set of neighbors of v
- Node features
  - Social networks: user profiles, user posts
  - 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 deep (fully connected) NN

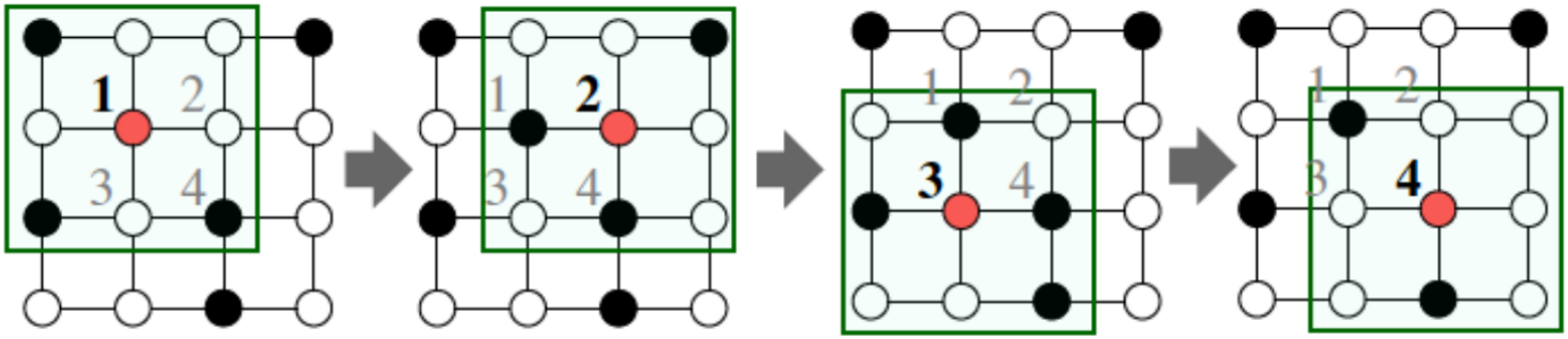


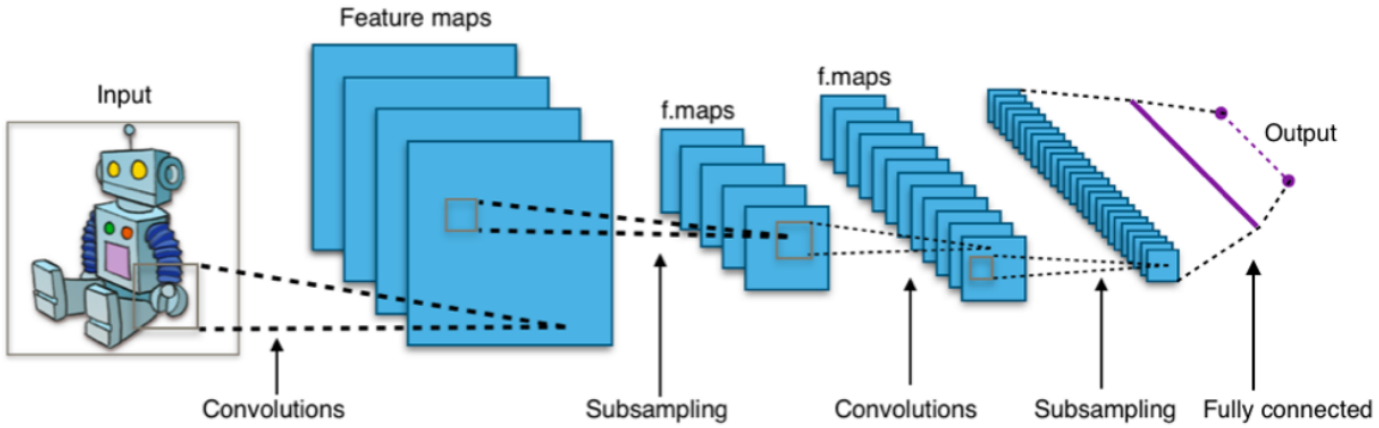
- Issues with this approach
  - $O(|V|)$ parameters
  - Not applicable to graphs of different sizes
  - Sensitive to node ordering

Idea from Convolutional Neural Nets

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

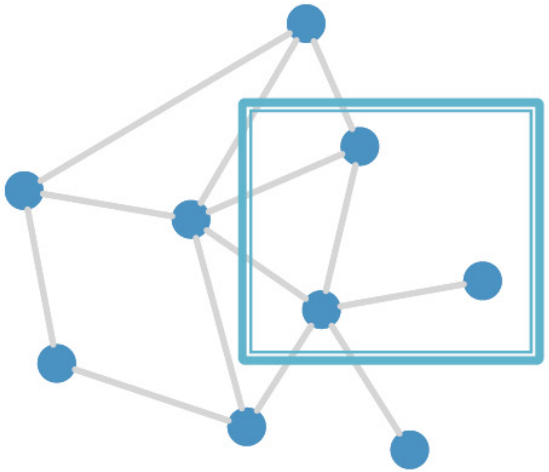
CNN on an image:



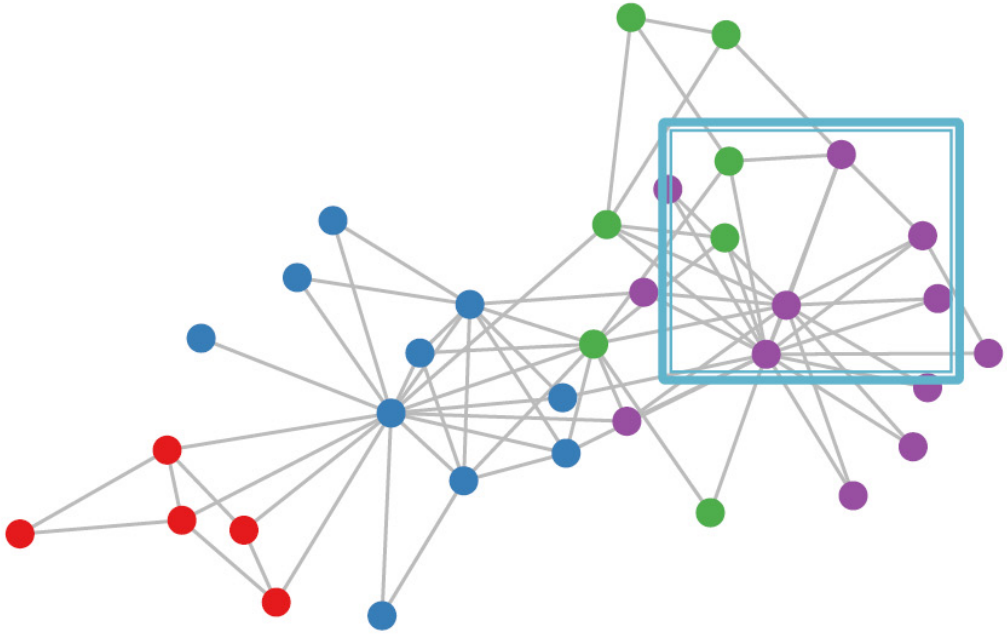


However, Real-world Graphs

But our graphs look like this:



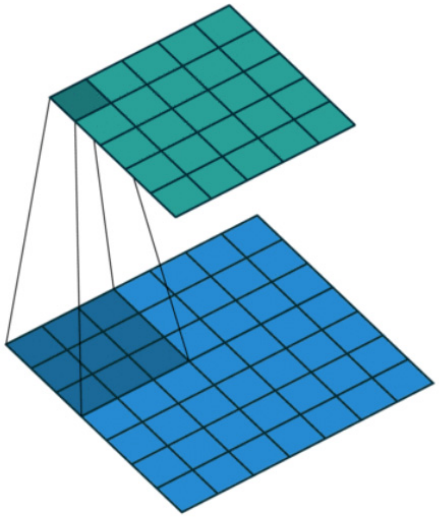
or this:



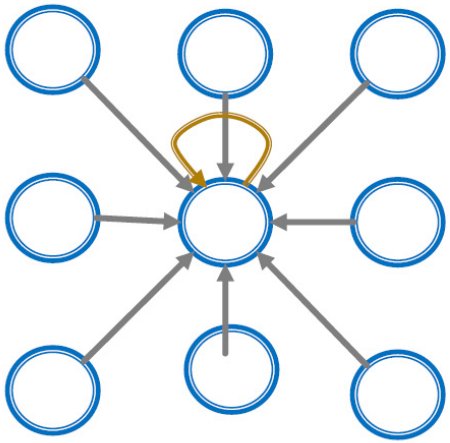
- No fixed notion of locality or sliding window on graphs
- Graph is permutation invariant

From Images to Graphs

- Convolutional neural net (CNN) layer with 3x3 filter
- Idea: transform information of neighbors & combine it
  - Transform "messages" h_i from neighbors: $W_i h_i$
  - Add them up: $\sum_i W_i h_i$



Image

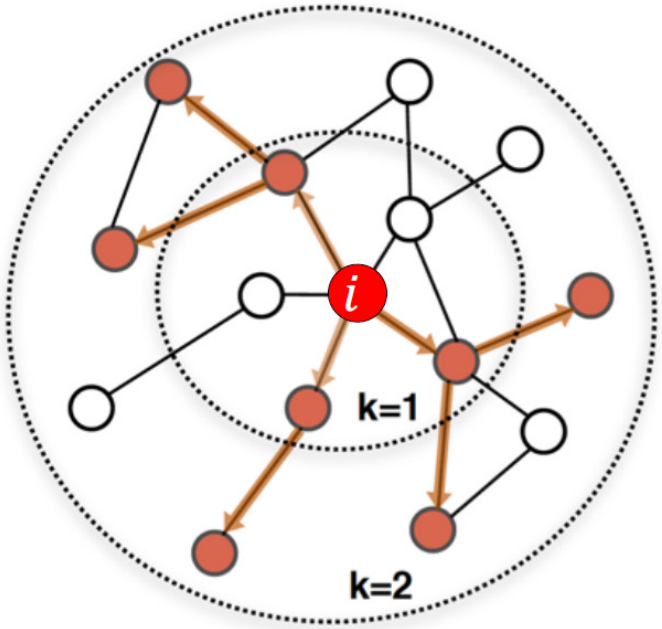


Graph

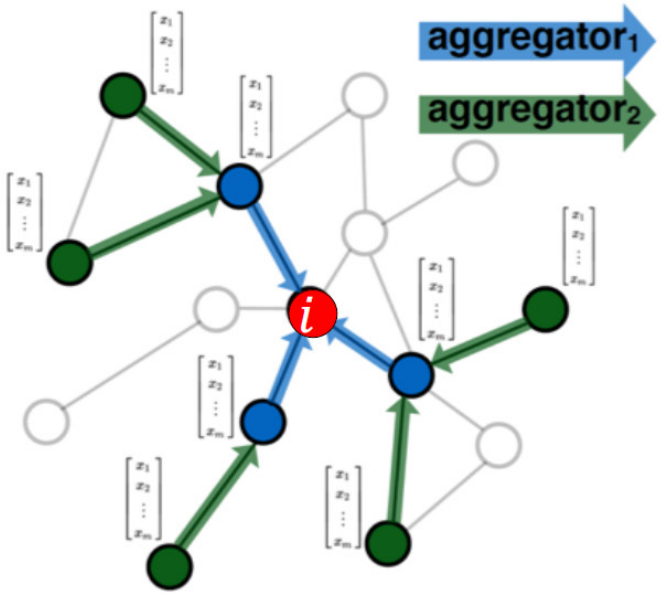
Graph Neural Networks

- Idea: Node's neighborhood defines a computation graph

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



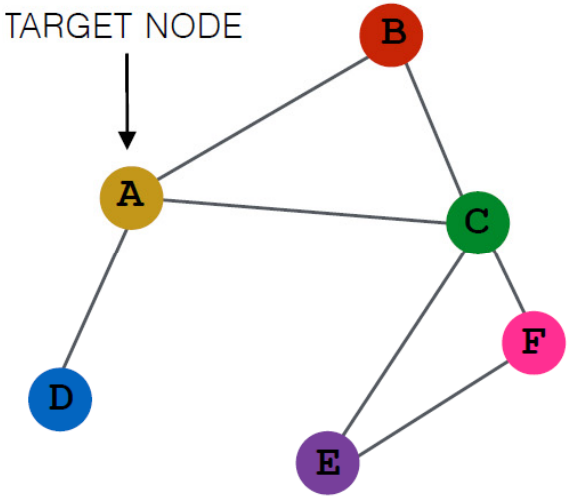
Determine node computation graph



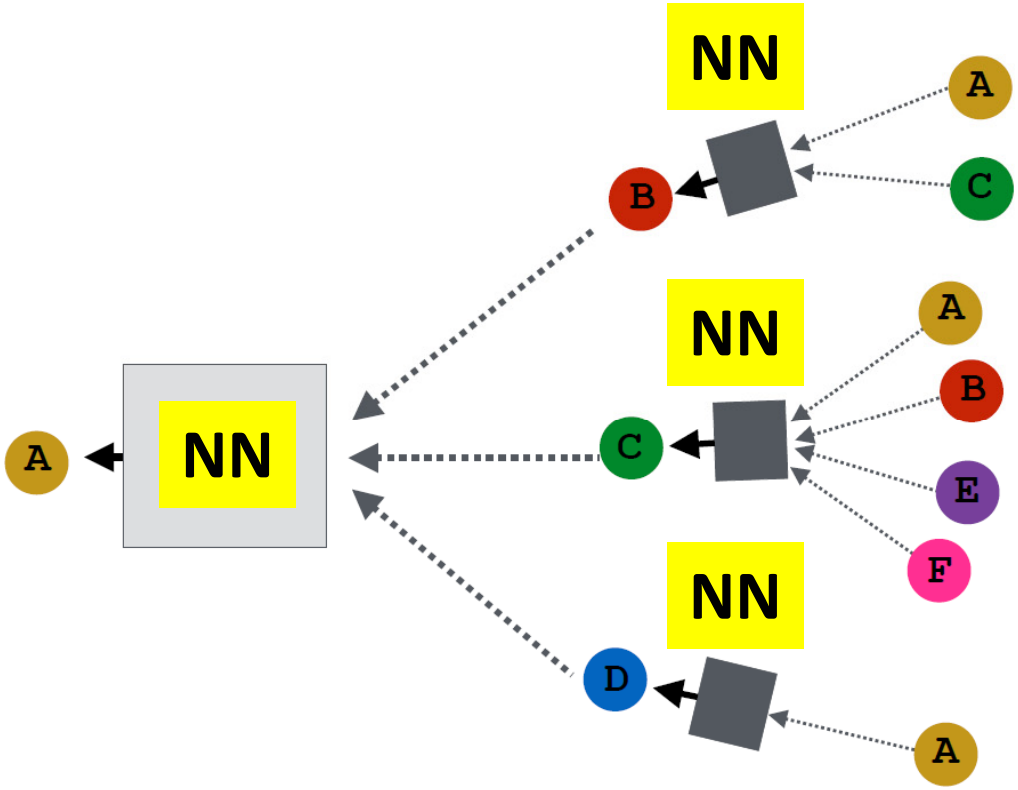
Propagate and transform information

Neighborhood Aggregation

- Intuition: Generate node embeddings based on local network neighborhoods
- Nodes aggregate info from their neighbors using NN



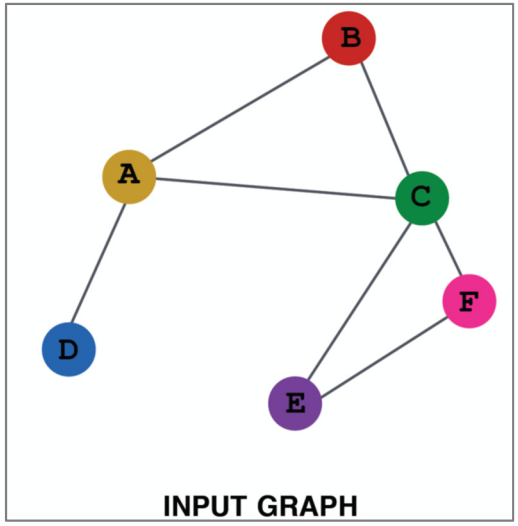
INPUT GRAPH

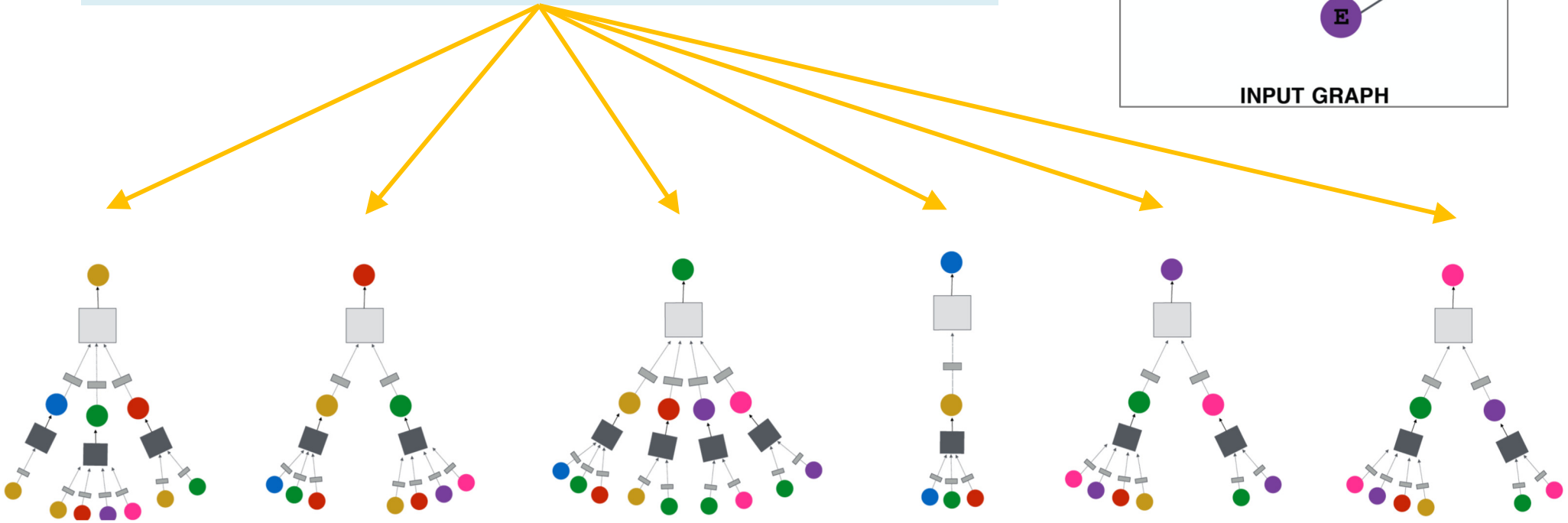


Neighborhood Aggregation

- Network neighborhood defines a computation graph

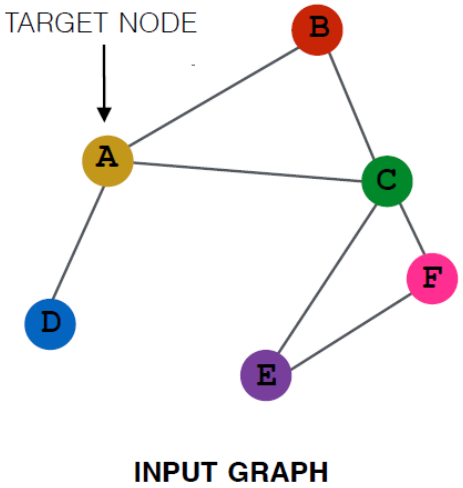
Every node defines a computation graph based on its neighborhood!



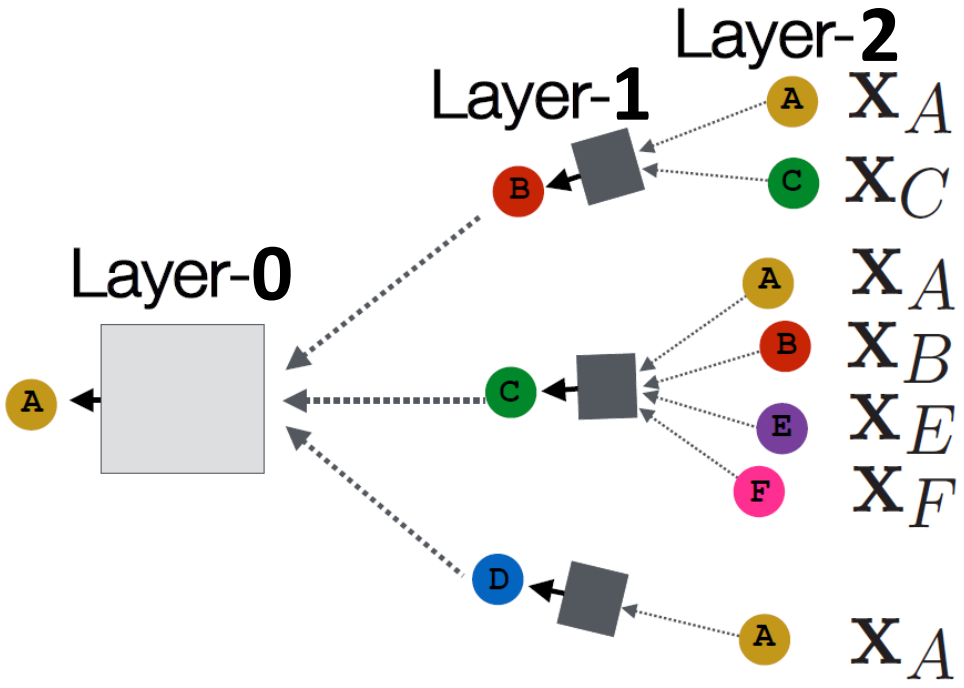


Deep Model: Multiple Layers

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



INPUT GRAPH



Neighborhood Aggregation

- Average neighbor messages and apply a NN

Initial 0-th layer embeddings are equal to node features

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

embedding of v at layer l

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

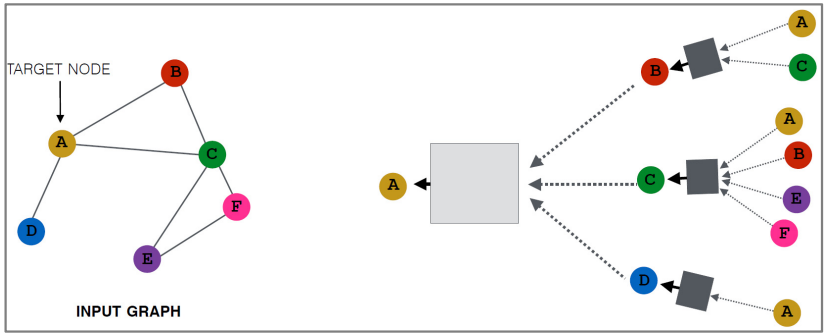
$$\mathrm{z}_v = \mathrm{h}_v^{(L)}$$

Embedding after L layers of neighborhood aggregation

Non-linearity (e.g., ReLU)

Average of neighbor's previous layer embeddings

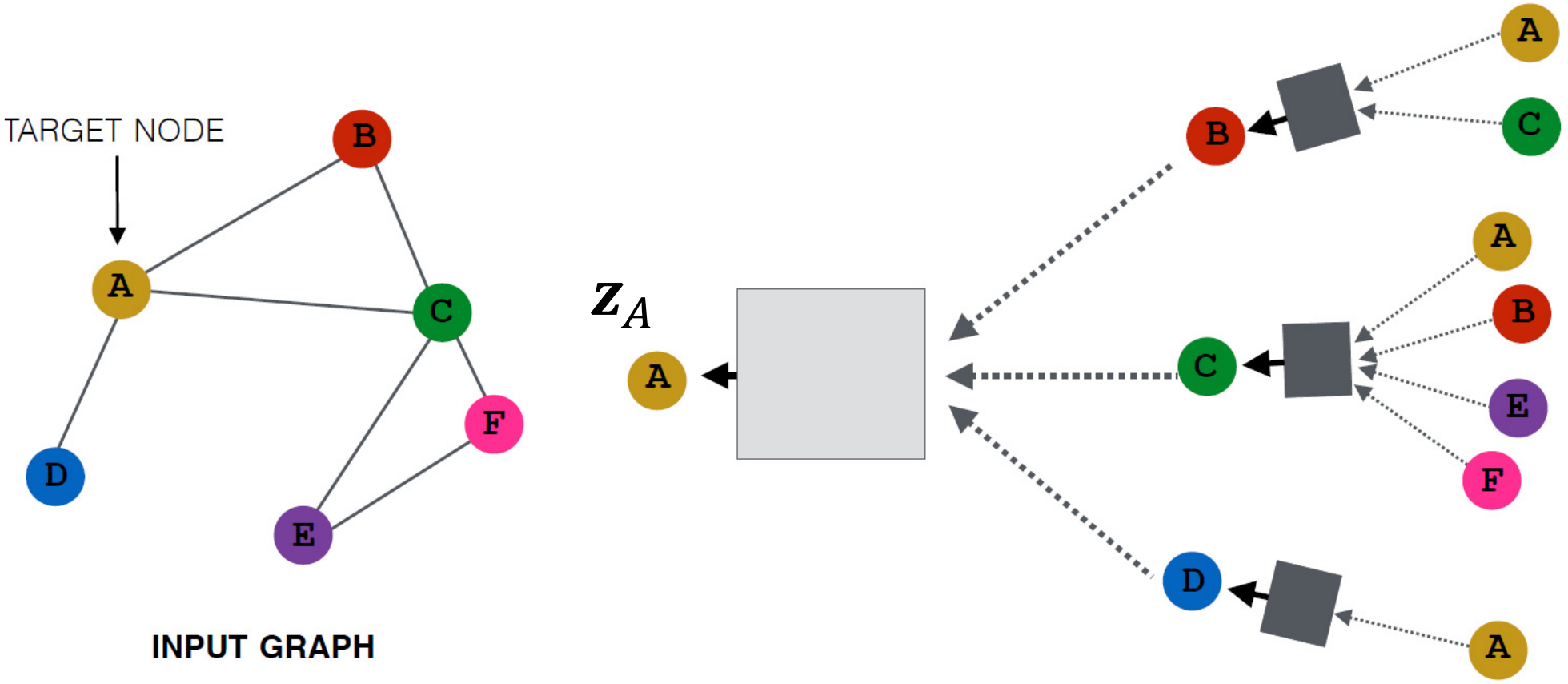
Total number of layers



Training the GNN Model

How do we train the model to generate embeddings?

Need to define a loss function on the embeddings!!



Model Parameters

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

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

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

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

Final node embedding

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

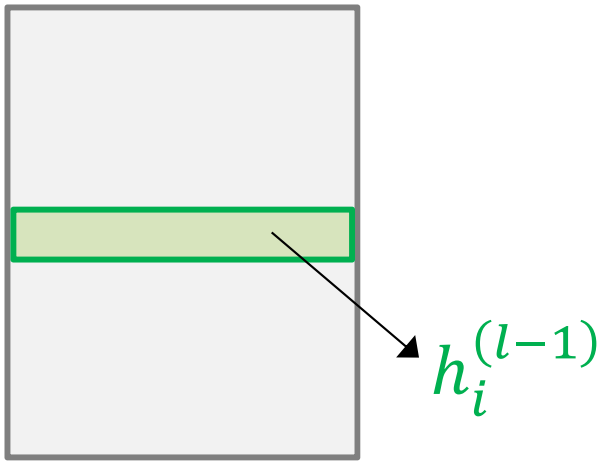
- h_v^l: the hidden representation of node v at layer l
- W_l: weight matrix for neighborhood aggregation at layer l
- B_l: weight matrix for transforming hidden vector of self

Matrix Formulation

- Many aggregations can be performed efficiently by (sparse) matrix operations
- Let $H^{(l)} = \left[h_1^{(l)}, h_2^{(l)}, \ldots, h_{|V|}^{(l)}\right]^{\mathrm{T}}$
- Then: $\sum_{u \in N(v)} h_u^{(l)} = A_{v,:} H^{(l)}$
- Let D be the diagonal matrix, where $D_{v,v} = \mathrm{Deg}(v) = |N(v)|$
  - Inverse of D: D^{-1} is also diagonal $D_{v,v}^{-1} = 1/N(v)$
- Therefore:

$$\sum_{u \in N(v)} \frac{h_u^{(l-1)}}{|N(v)|} \Rightarrow H^{(l+1)} = D^{-1} A H^{(l)}$$

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

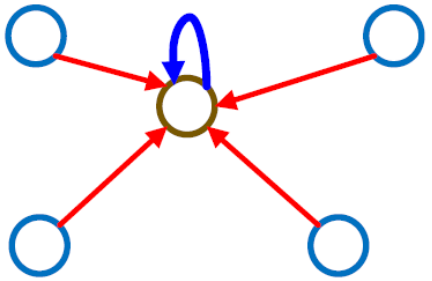


Matrix Formulation

- Re-write the update function in matrix form

$$H^{(l+1)} = \sigma\left(\tilde{A}H^{(l)}W_l^{\mathrm{T}} + H^{(l)}B_l^{\mathrm{T}}\right)$$

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



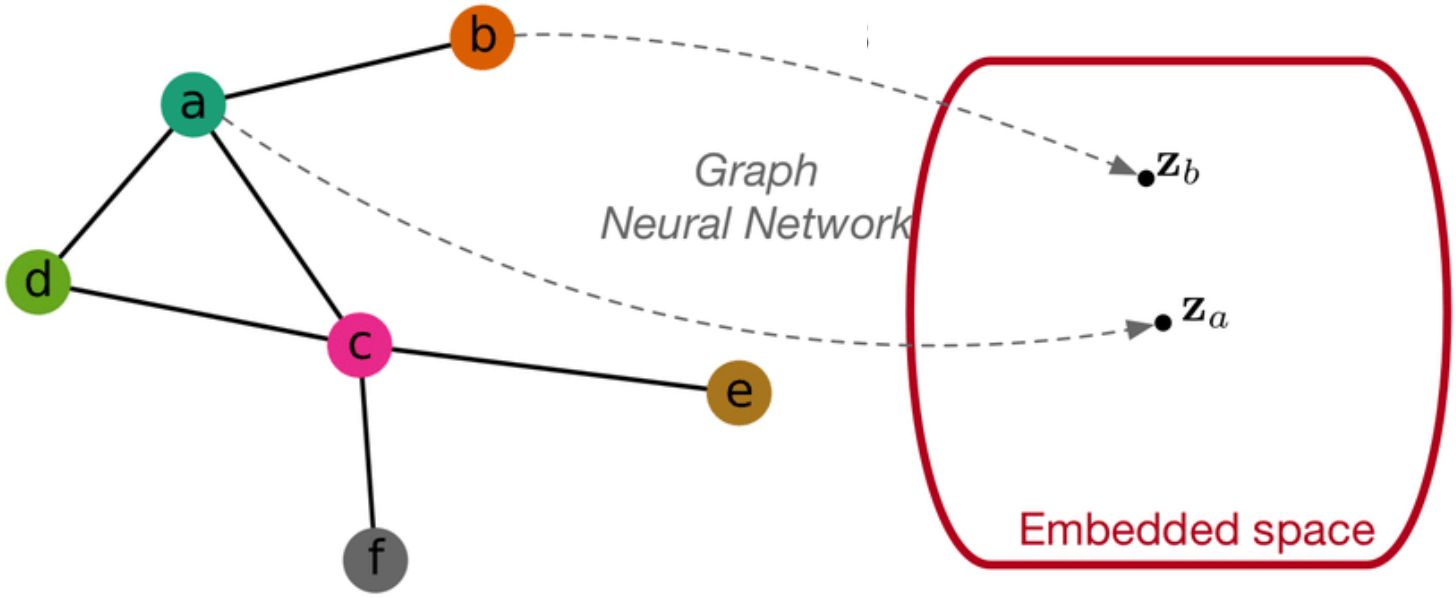
$$H^{(l)} = [h_1^{(l)} \dots h_{|V|}^{(l)}]^T$$

  - 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

GNN Training

- Node embedding $\boldsymbol{z}_v$ is a function of input graph
- **Supervised** learning: $\min_{\Theta} \mathcal{L}\left(\boldsymbol{y}, f\left(\boldsymbol{z}_v\right)\right)$
  - $\mathcal{L}$: L2 norm (y is real-value) or cross entropy (y is categorical)
- **Unsupervised** learning (no node label available)
  - Use graph structure as the supervision



Node classification:

$\mathrm{softmax}(\mathbf{z_n})$

e.g. Kipf & Welling (ICLR 2017)

Graph classification:

$\mathrm{softmax}(\sum_n \mathbf{z_n})$

e.g. Duvenaud et al. (NIPS 2015)

Link prediction:

$p(A_{ij}) = \sigma(\mathbf{z_i^T z_j})$

Kipf & Welling (NIPS BDL 2016)

"Graph Auto-Encoders"

Unsupervised GNN Training

- "Similar" nodes have similar embeddings

$$\mathcal{L} = \sum_{\mathbf{z}_u, \mathbf{z}_v} \mathrm{CE}\left(y_{u,v}, \mathrm{DEC}(\mathbf{z}_u, \mathbf{z}_v)\right)$$

  - $y_{u,v} = 1$ when nodes u and v are similar
  - CE: cross entropy
  - DEC: the decoder such as *inner product*
- Node similarity is what GRL aims to preserve
  - Random walks (e.g, node2vec, DeepWalk)
  - Matrix Factorization
  - Node proximity (e.g., Jaccard, Adamic-Adar)

Supervised GNN Training

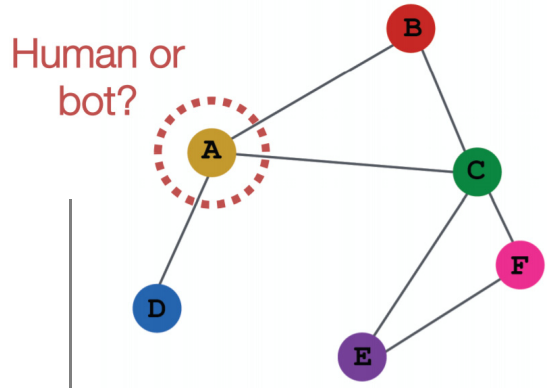
- Directly train the model for a supervised task (e.g., node classification)
  - Use cross entropy

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

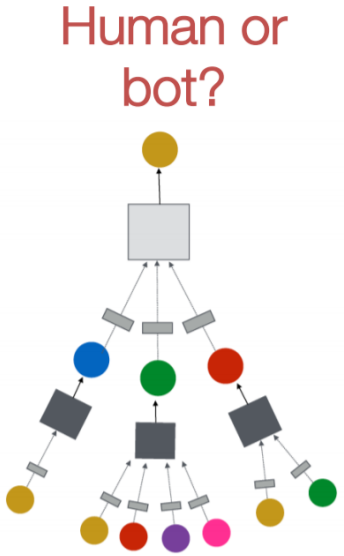
Encoder output: node embedding

Classification weights

Node class label

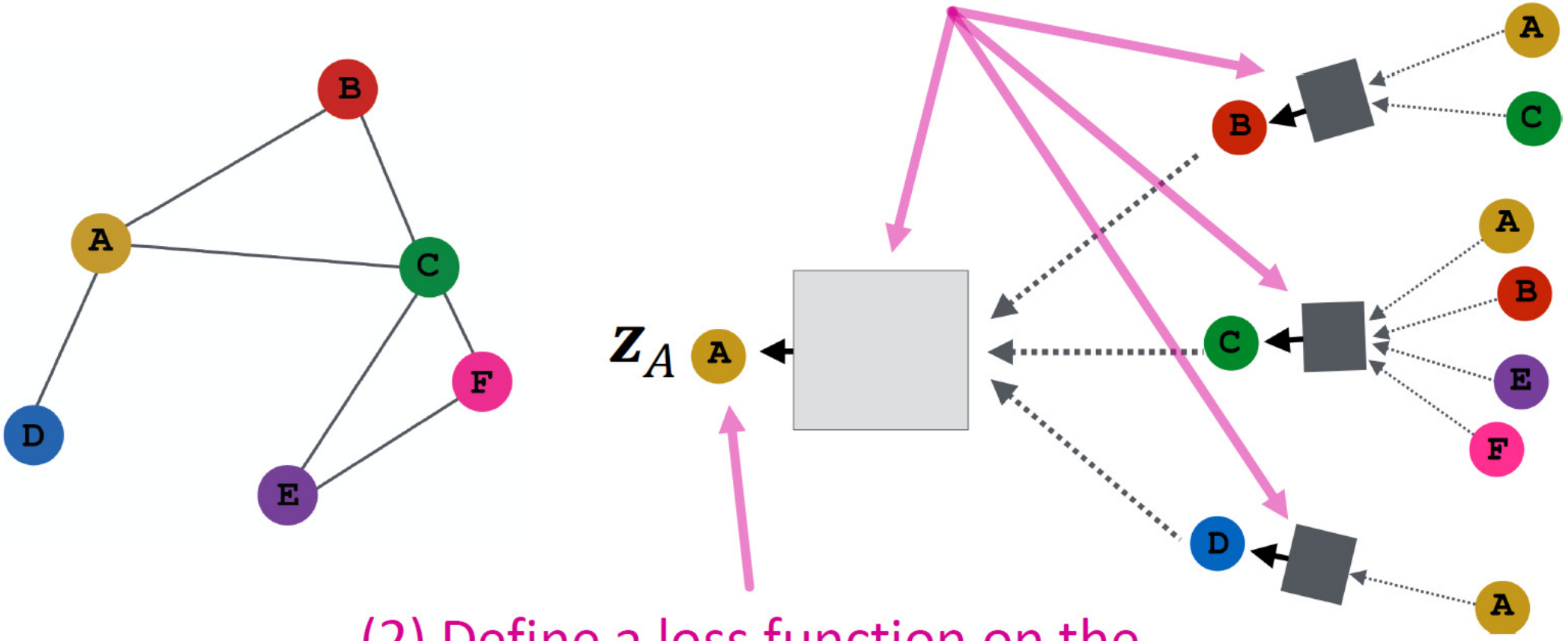


e.g., an online social network



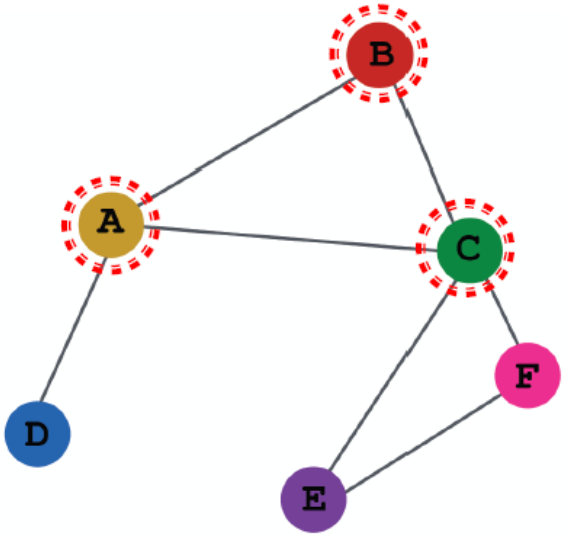
GNN Model Design

(1) Define a neighborhood aggregation function



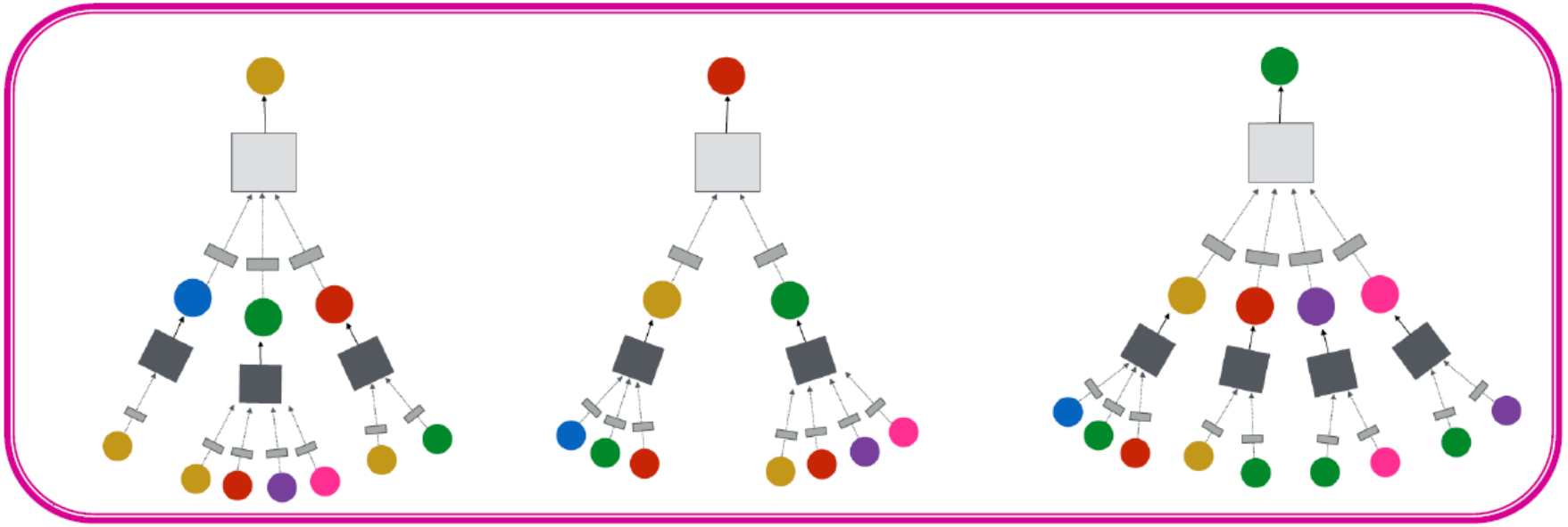
(2) Define a loss function on the embeddings

GNN Model Design

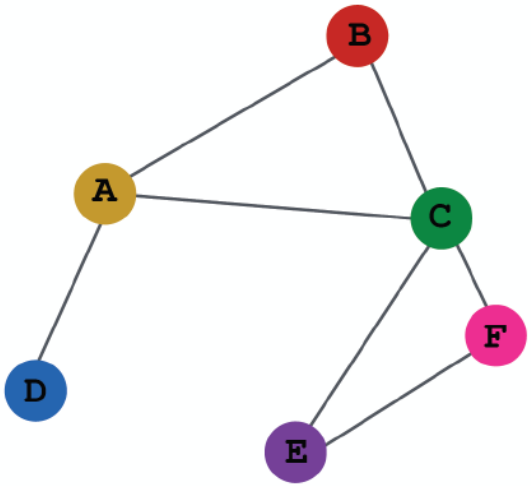


INPUT GRAPH

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



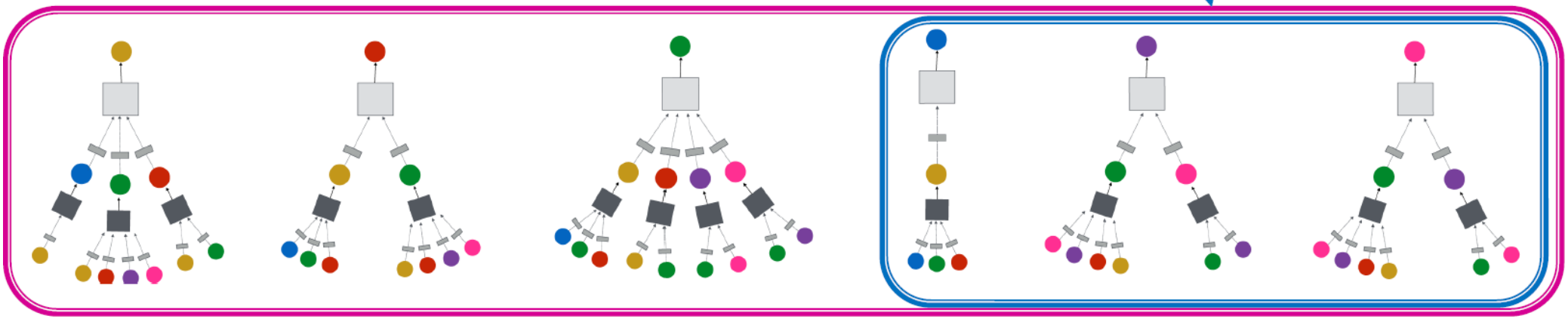
GNN Model Design



INPUT GRAPH

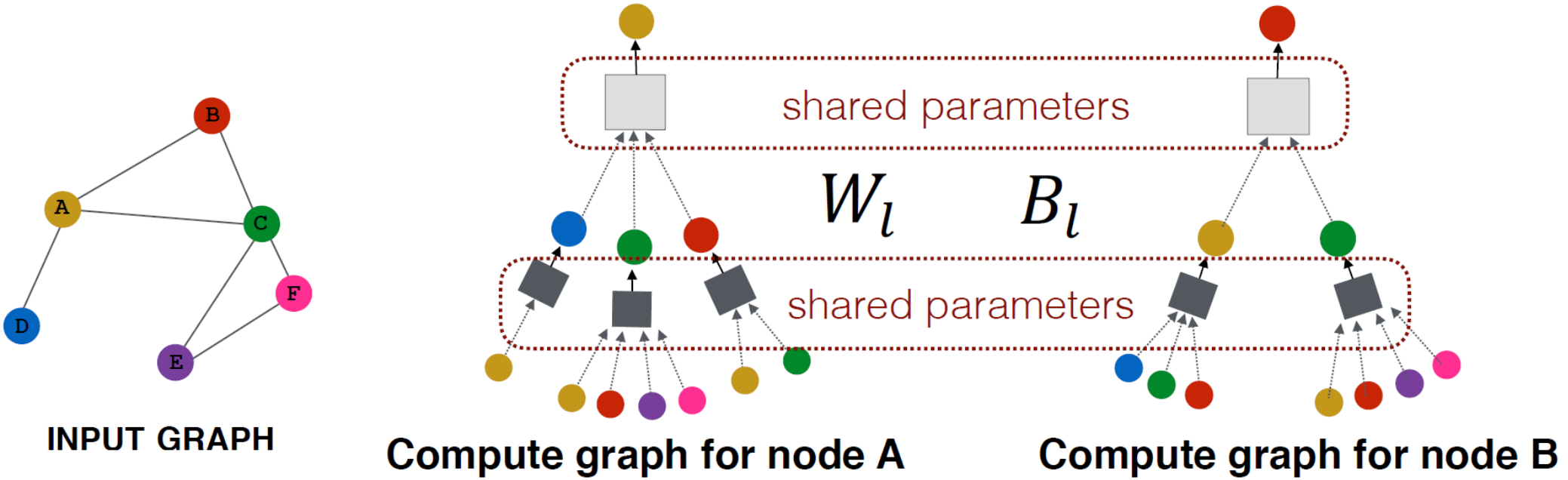
(4) Generate embeddings for nodes as needed

Even for nodes we never trained on!

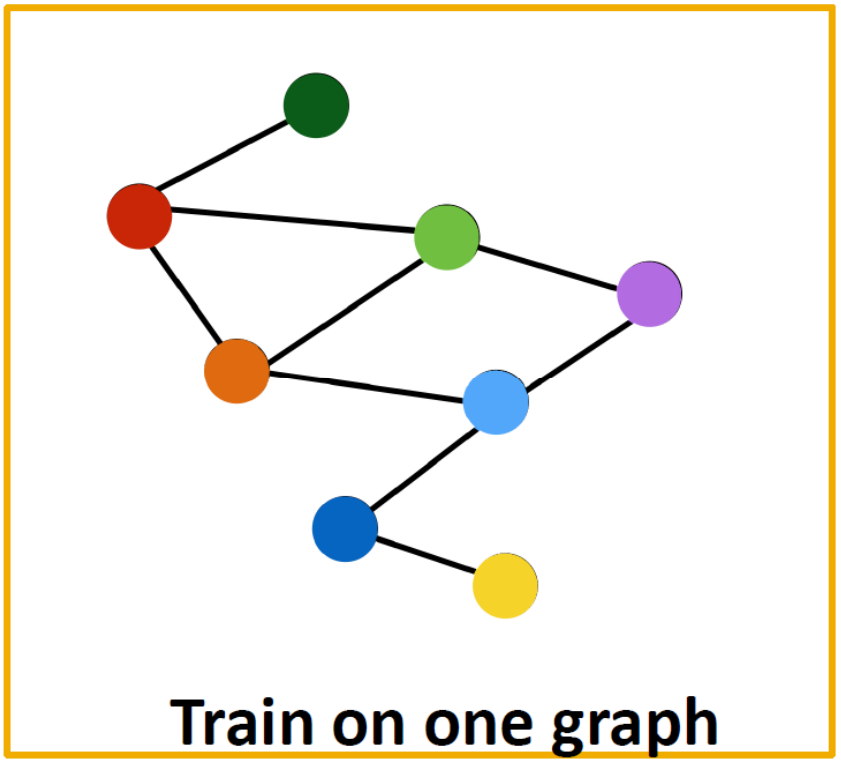


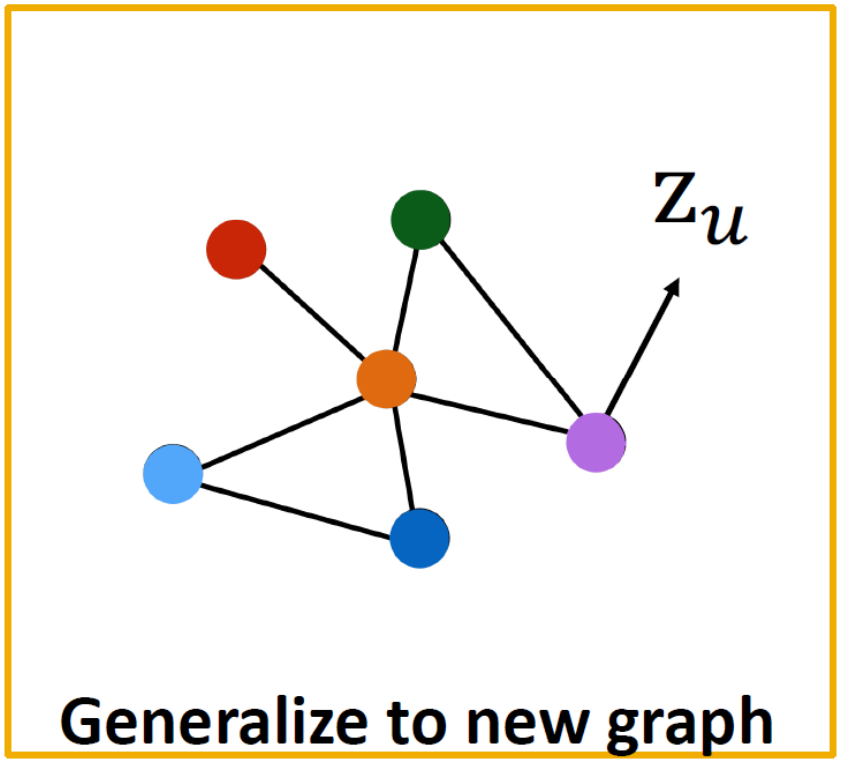
Inductivity Capability

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



Inductivity Capability: **New Graphs**



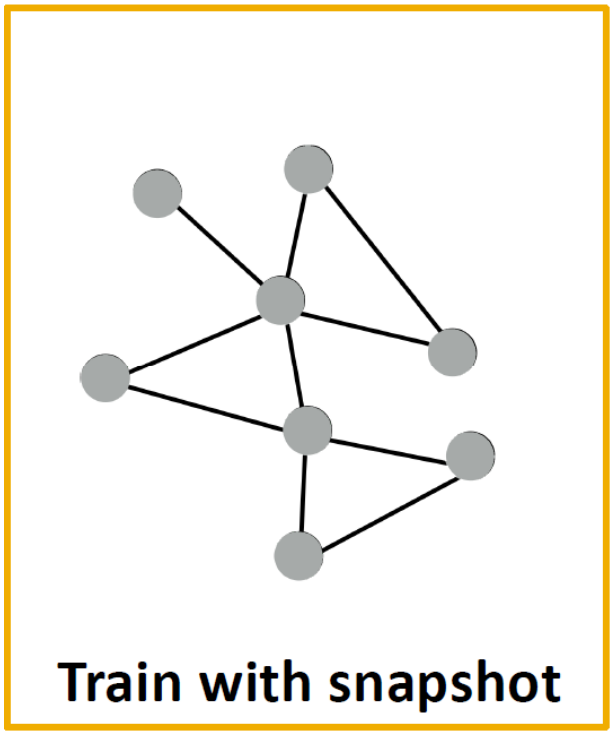


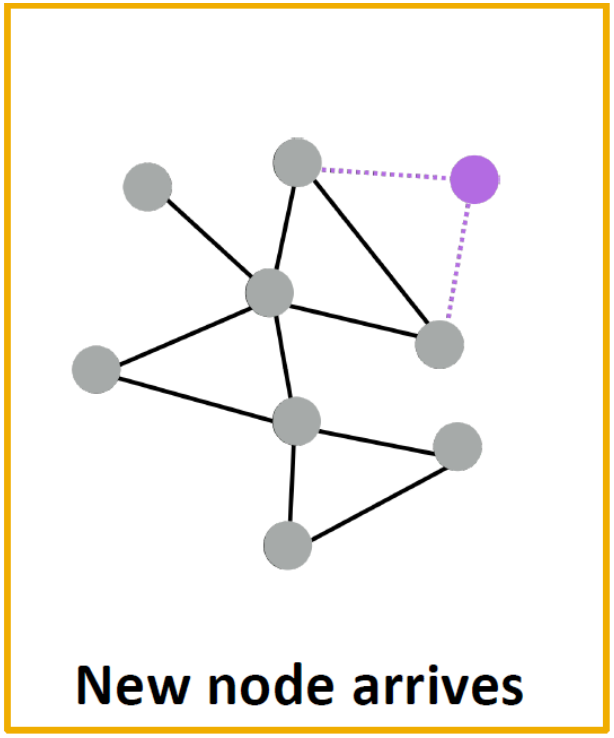
Inductive node embedding learning

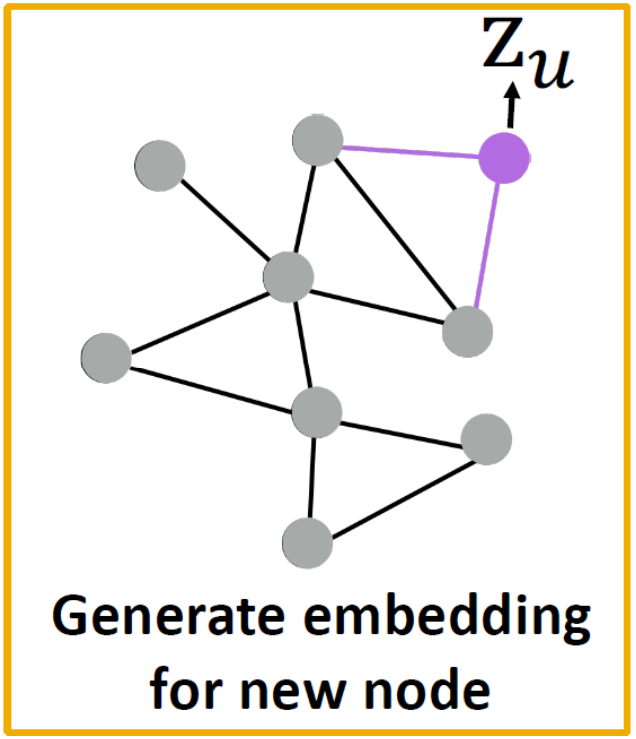
Transfer/Generalize to entirely unseen graphs

E.g., train on social network from Twitter and generate embeddings for nodes on newly collected data about Instagram

Inductivity Capability: **New Nodes**







- Many application settings constantly encounter previously unseen nodes
  - E.g., Instagram, YouTube, Facebook
- Need to generate new embedding "**on the fly**"

不經過某種額外訓練而直接進行產生embeddings

Quick Recap

- Recap: generate node embeddings by aggregating neighborhood information
  - Allow for parameter sharing in the encoder
  - Allow for inductive learning
- We saw a **basic form of GNN idea**
  - Key distinctions are in how different approaches aggregate information across the layers
  - We will cover some state-of-the-art variants