Graph Neural Networks: Model I

Jiaxuan You

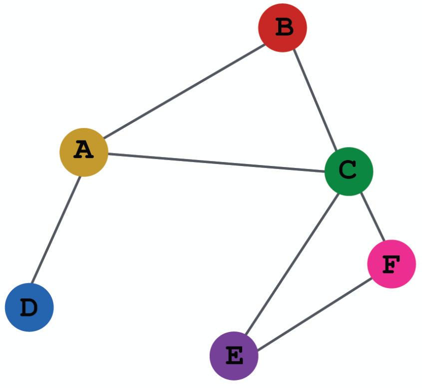
Assistant Professor at UIUC CDS

CS598: Deep Learning with Graphs, 2024 Fall

https://ulab-uiuc.github.io/CS598/

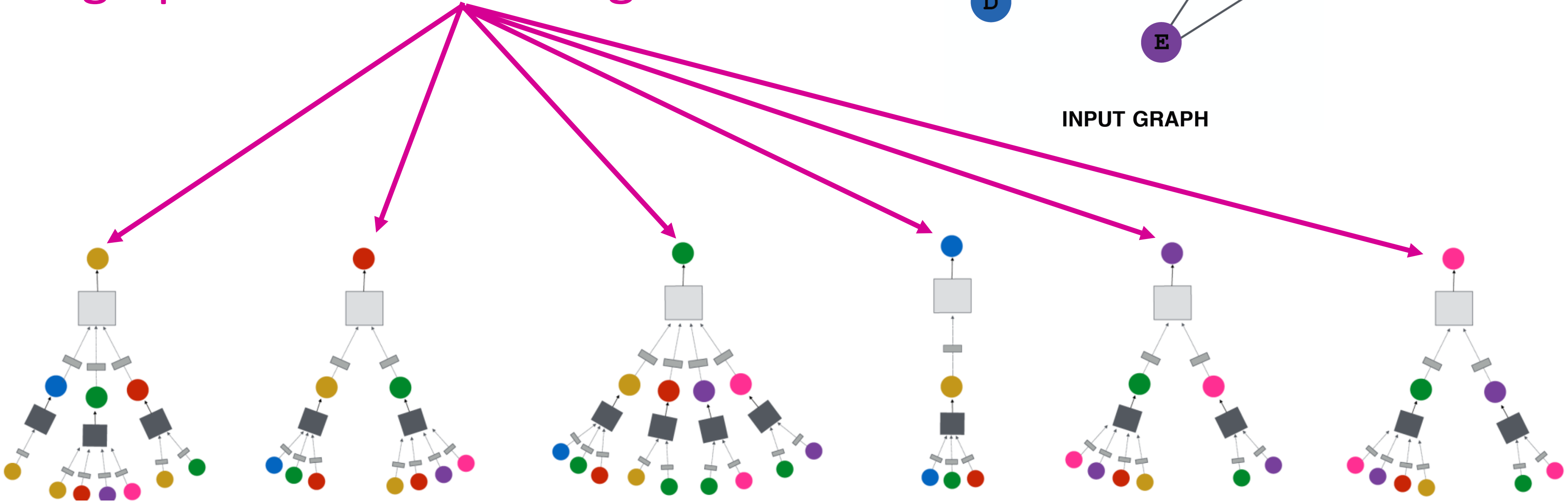
Recap: GNN Defines Comp Graph for Each Node

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



INPUT GRAPH

Every node defines a computation graph based on its neighborhood!



Recap: GCN Encoder

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

Initial 0-th layer embeddings are equal to node features

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

embedding of v at layer k

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

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

Average of neighbor's previous layer embeddings

Total number of layers

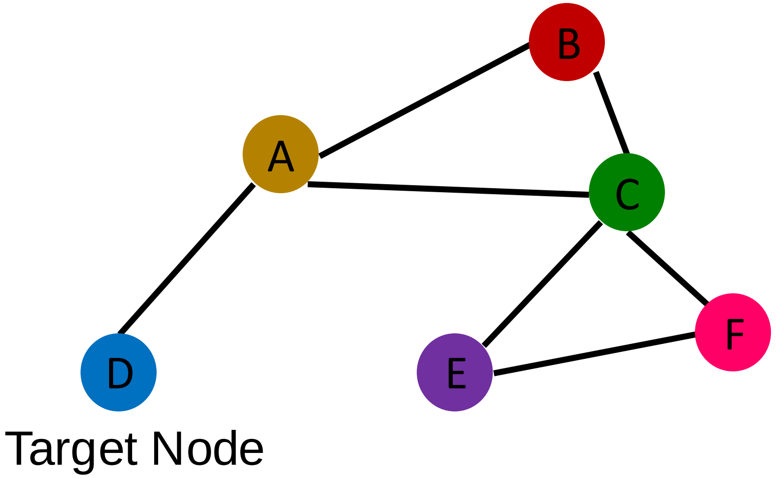
Embedding after K layers of neighborhood aggregation

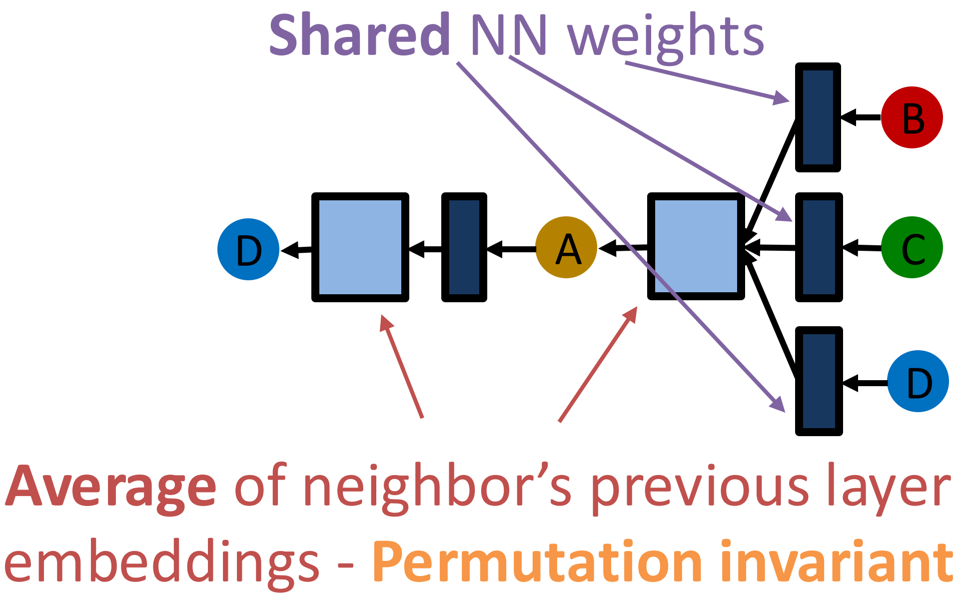
Non-linearity (e.g., ReLU)

GCN: Invariance and Equivariance

What are the invariance and equivariance properties for a GCN?

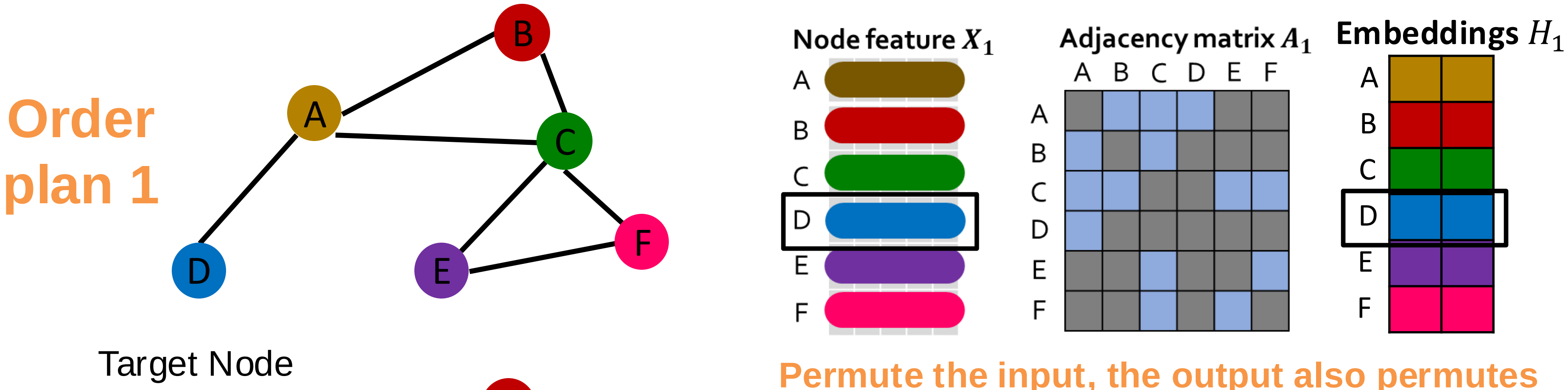
- **Given a node**, the GCN that computes its embedding is **permutation invariant (output one embedding)**



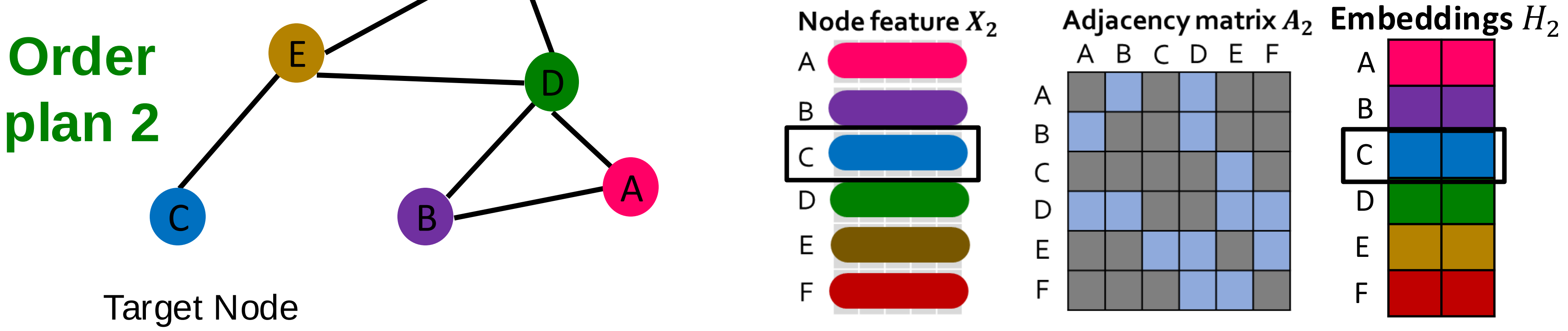


GCN: Invariance and Equivariance

- **Considering all nodes in a graph**, GCN computation is **permutation equivariant (output multiple embeddings)**



Permute the input, the output also permutes accordingly - **permutation equivariant**



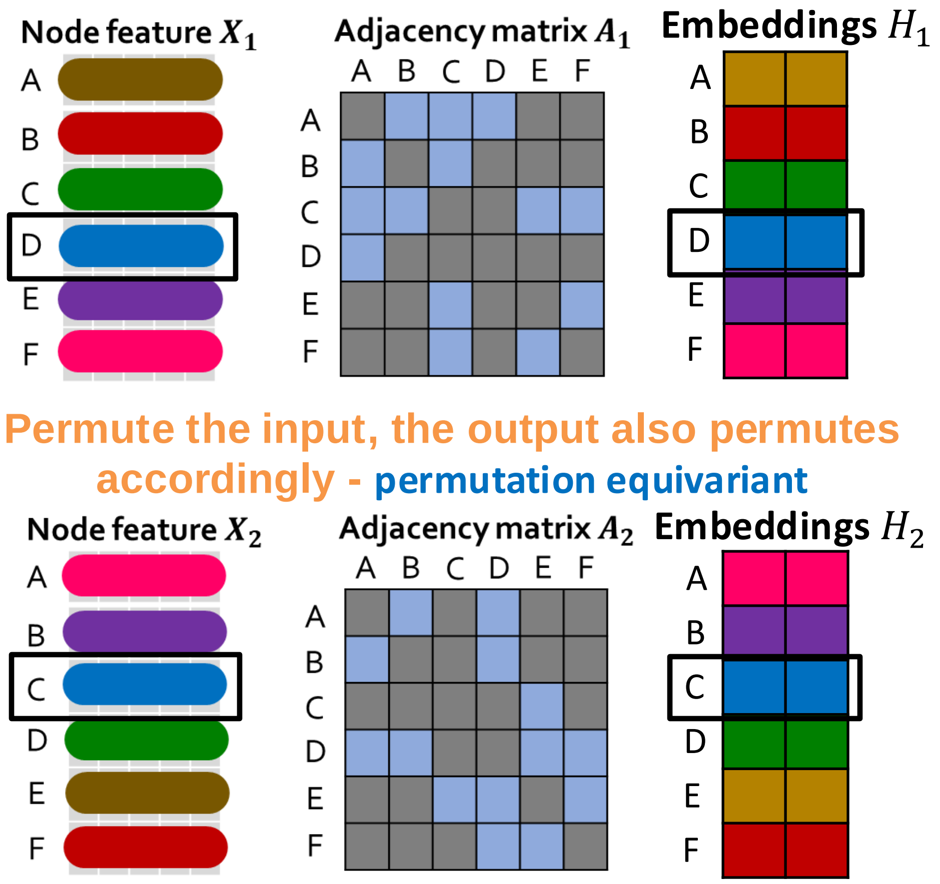
GCN: Invariance and Equivariance

- **Considering all nodes in a graph**, GCN computation is **permutation equivariant**

Detailed reasoning:

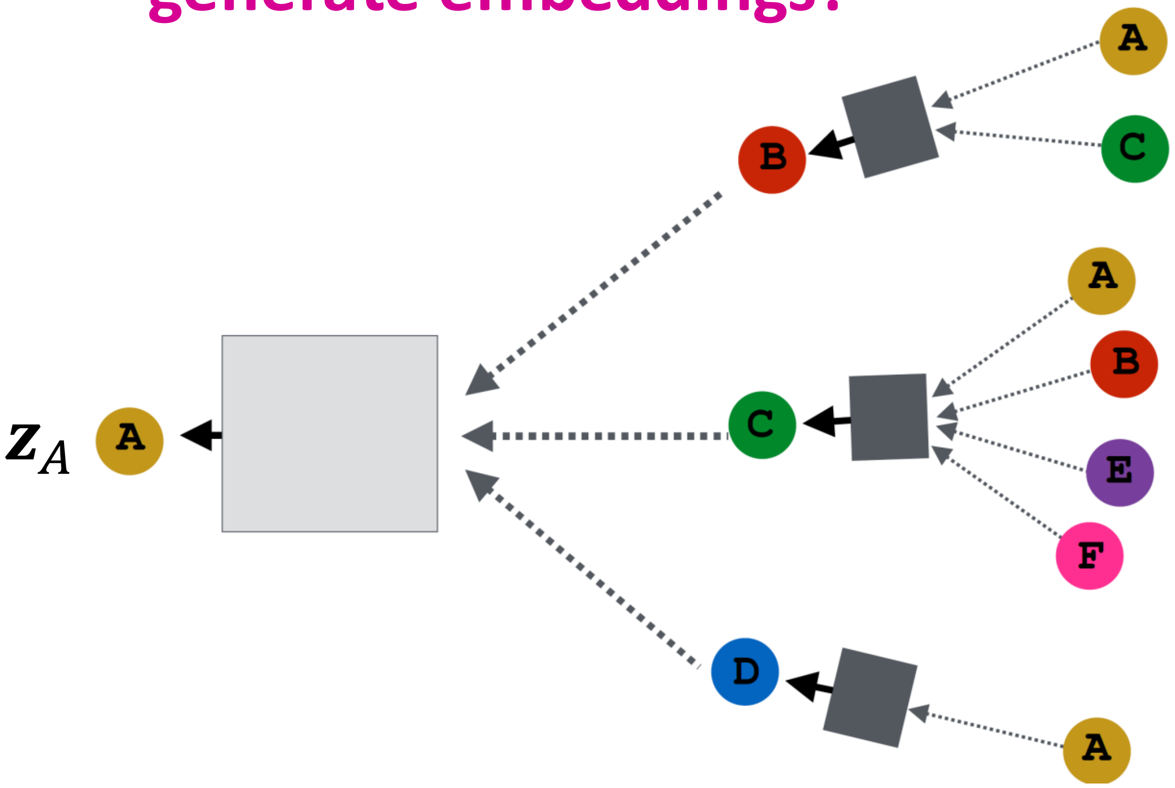
1. The rows of **input node features** and **output embeddings** are **aligned**
2. We know computing the embedding of **a given node** with GCN is **invariant.**
3. So, after permutation, the **location** of **a given node** in the **input node feature** matrix is changed, and the **the output embedding of a given node stays the same** (the colors of node feature and embedding are **matched**)

This is permutation equivariant



Training the GCN Model

How do we train the GCN to generate embeddings?



Need to define a loss function on the embeddings.

GCN Model Parameters

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

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

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

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

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

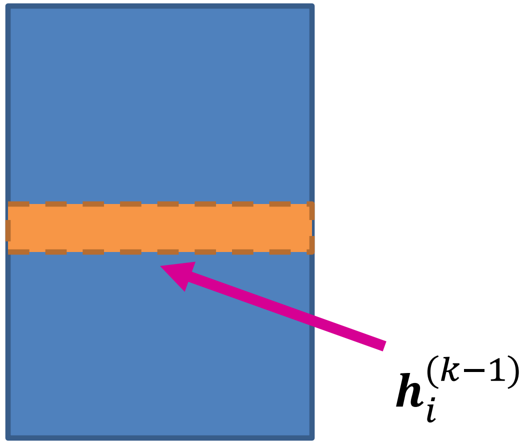
- W_k: weight matrix for neighborhood aggregation
- B_k: weight matrix for transforming hidden vector of self

GCN 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,**

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

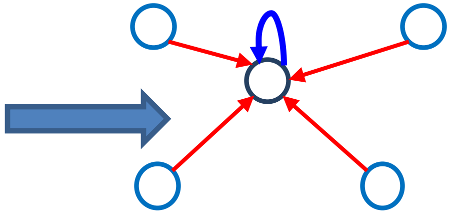
Matrix of hidden embeddings $H^{(k-1)}$



GCN 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

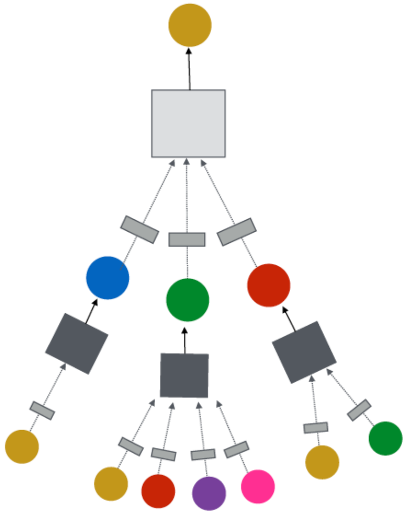
How to Train A GNN

- Objective: $\min_{\Theta} \mathcal{L}(\boldsymbol{y}, f(\boldsymbol{z}_v))$
  - $\boldsymbol{y}$: node/edge/graph label
  - $f(\boldsymbol{z}_v)$ could be node/edge/graph-level prediction head (lecture 2)
  - Θ are trainable GNN weights
  - $\mathcal{L}$ could be L2 if $\boldsymbol{y}$ is real number, or cross entropy if $\boldsymbol{y}$ is categorical
- **Supervised setting**:
  - $\boldsymbol{y}$ are external labels to graphs
- **Unsupervised setting:**
  - Use the graph structure as the supervision, e.g., similarity based loss function (lecture 3)

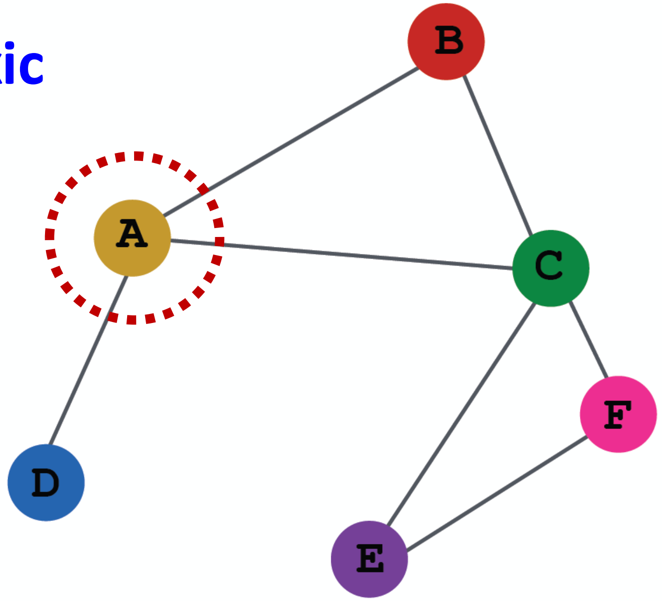
Example: Supervised Training

- **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

Example: Supervised Training

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

- Use cross entropy loss

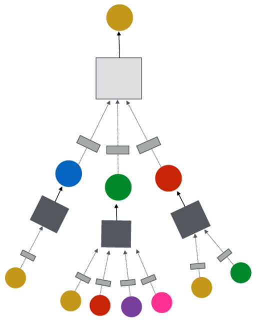
$$\mathcal{L} = -\sum_{v \in V} y_v \log(\sigma(\mathrm{z}_v^{\mathrm{T}}\theta)) + (1 - y_v)\log(1 - \sigma(\mathrm{z}_v^{\mathrm{T}}\theta))$$

Encoder output:
node embedding

Classification weights

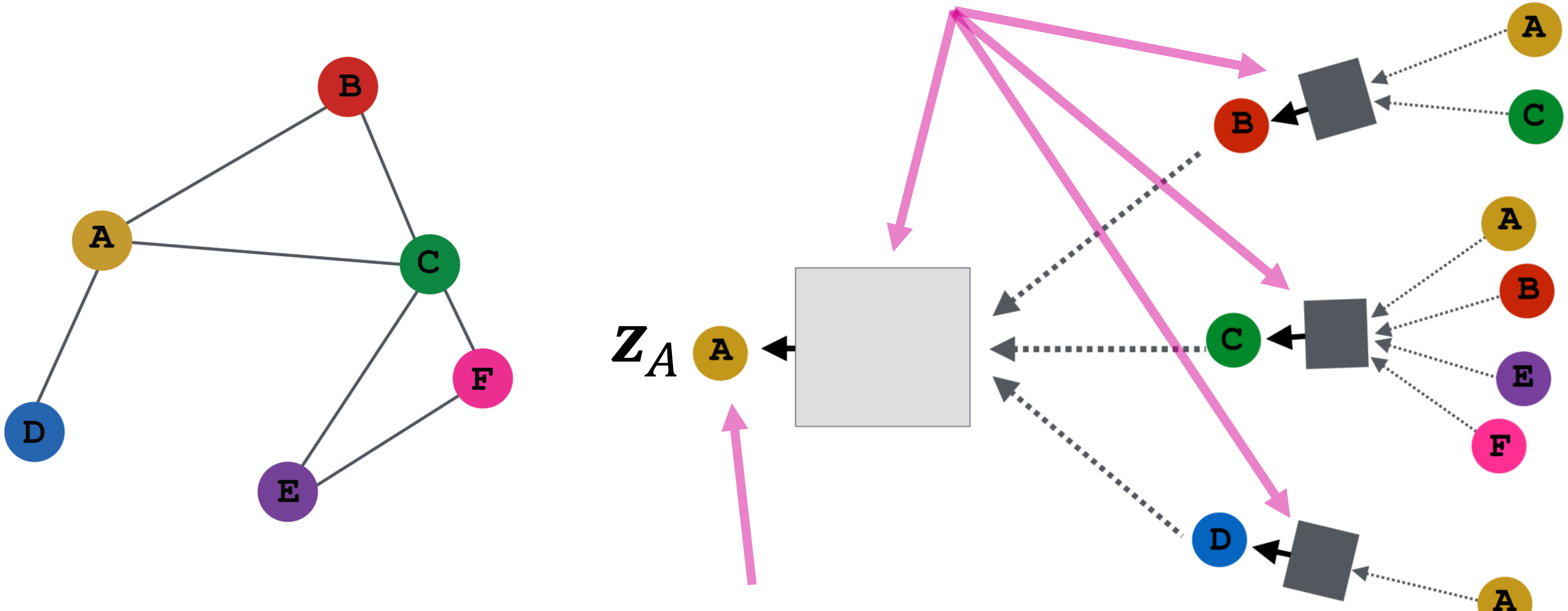
Node class label

Safe or toxic drug?



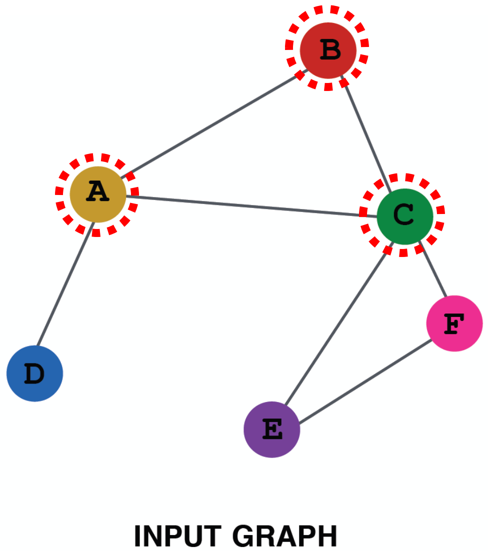
GCN Pipeline: Overview

(1) Define a neighborhood aggregation function

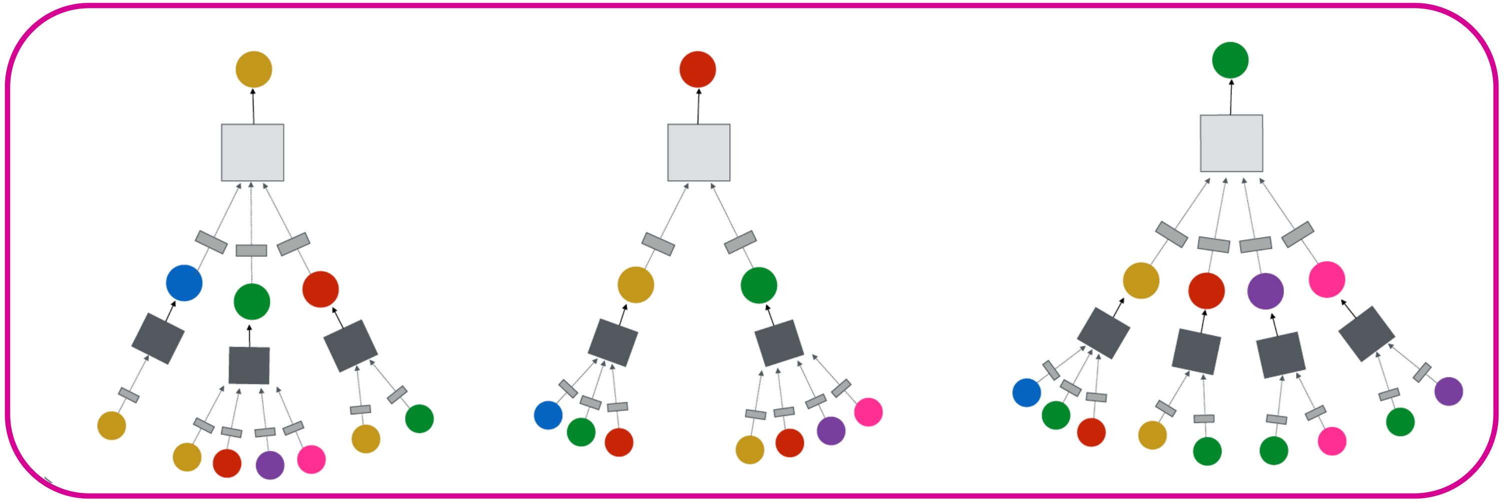


(2) Define a loss function on the embeddings

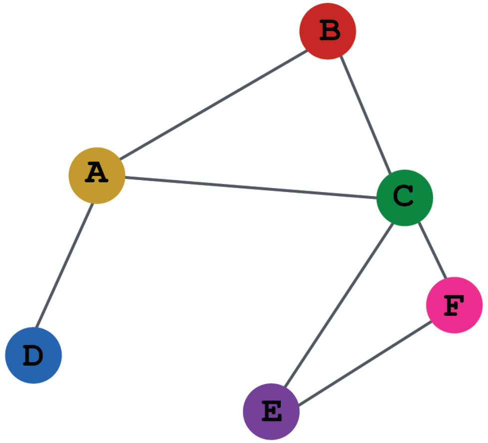
GCN Pipeline: Overview



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



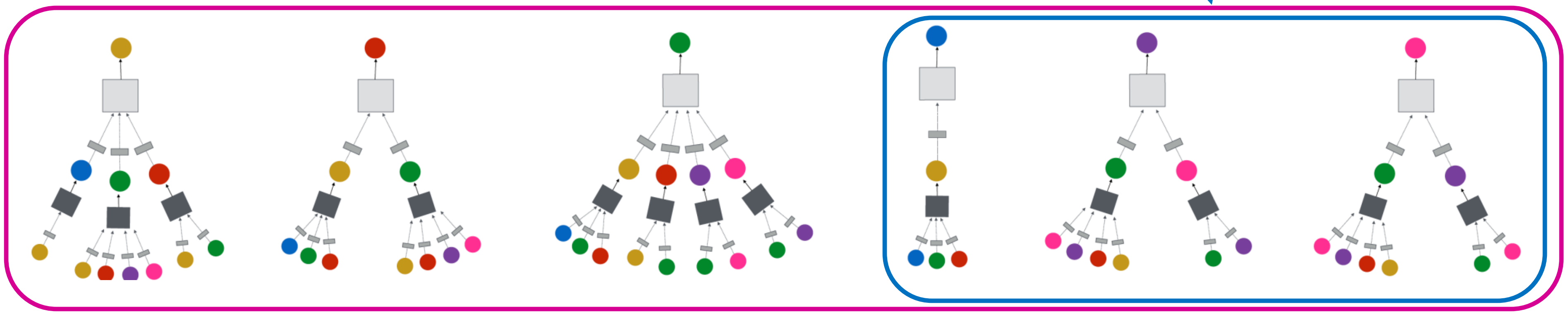
GCN Pipeline: Overview



INPUT GRAPH

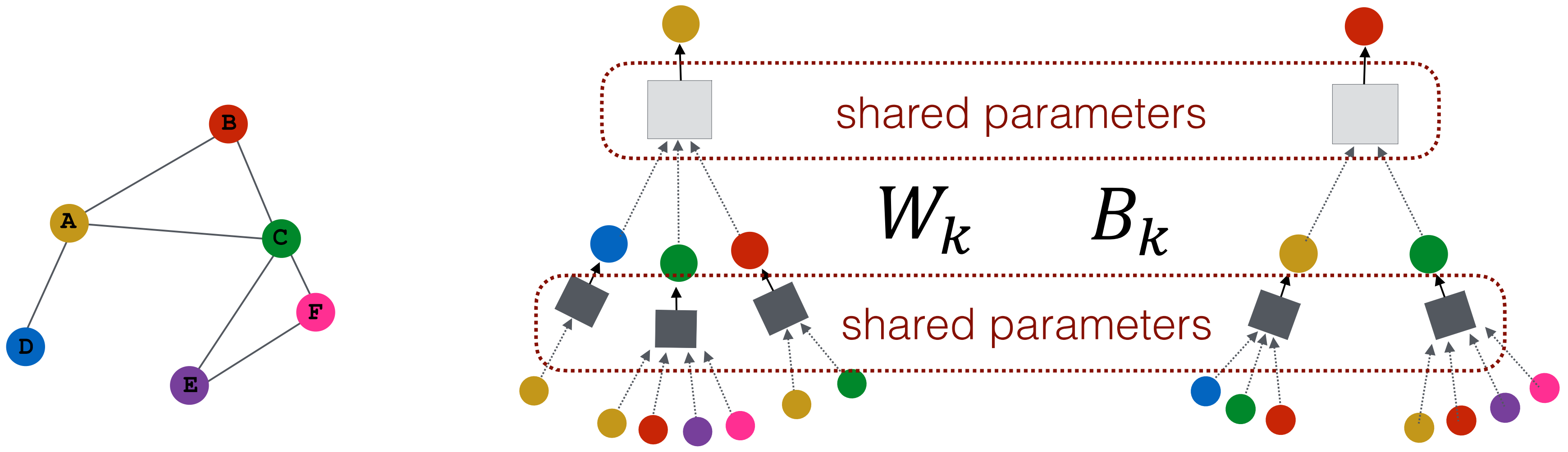
(4) Generate embeddings for nodes as needed

Even for nodes we never trained on!



GCN is Inductive – Can Generalize to New Data

- **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**!

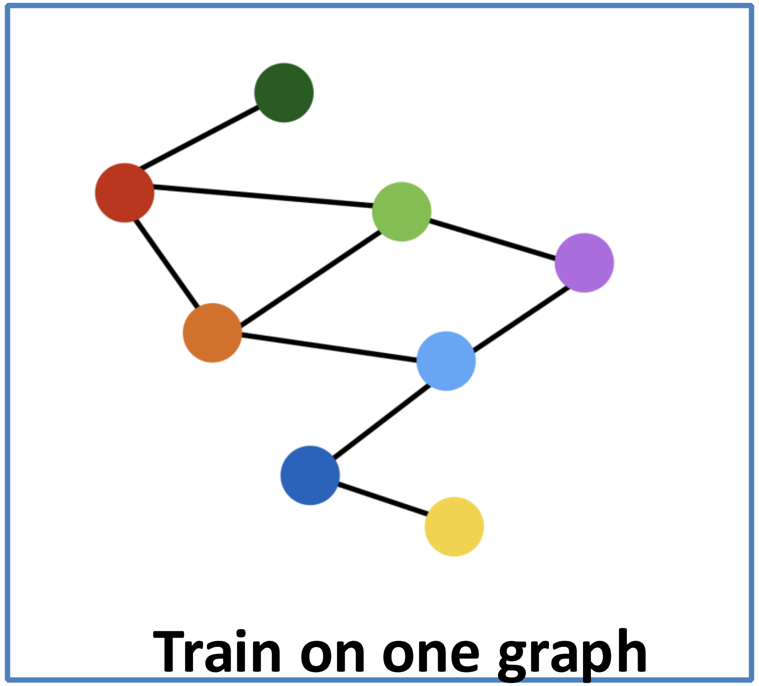


INPUT GRAPH

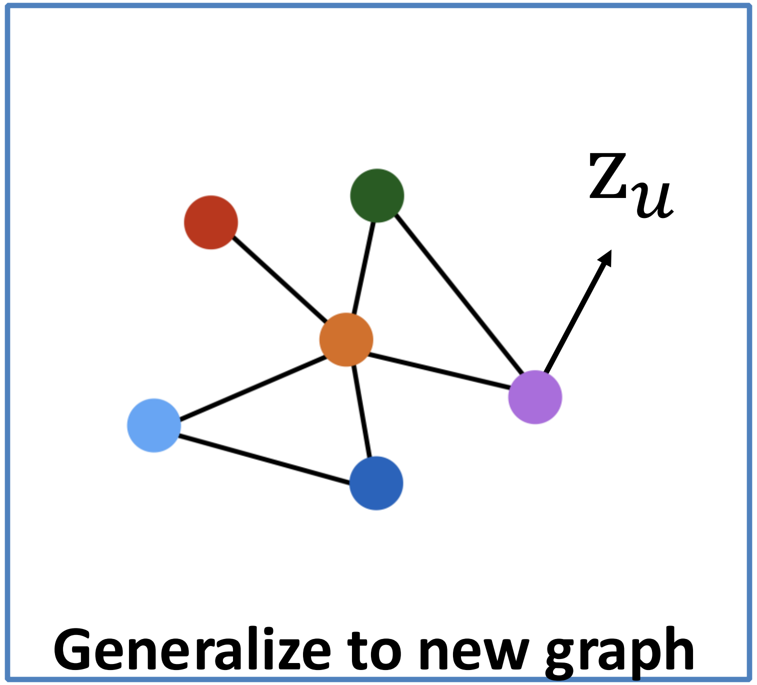
Compute graph for node A

Compute graph for node B

Inductive Capability: New Graphs



Train on one graph

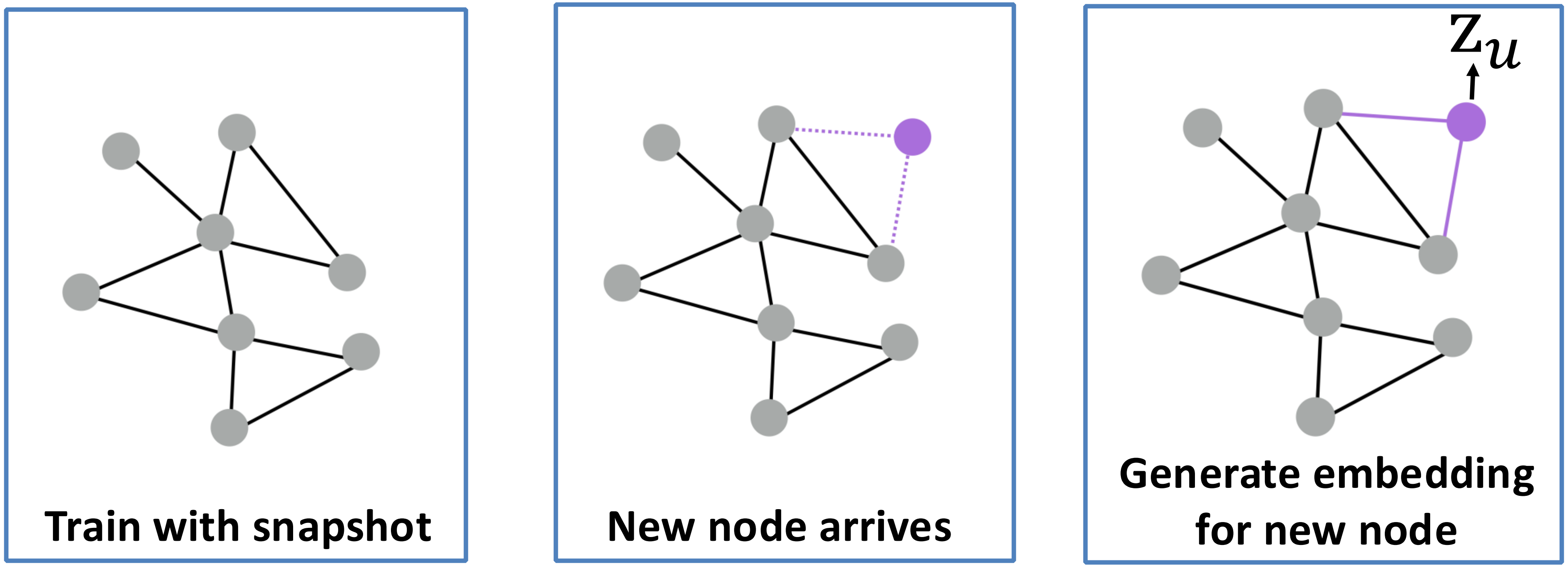


Generalize to new graph

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

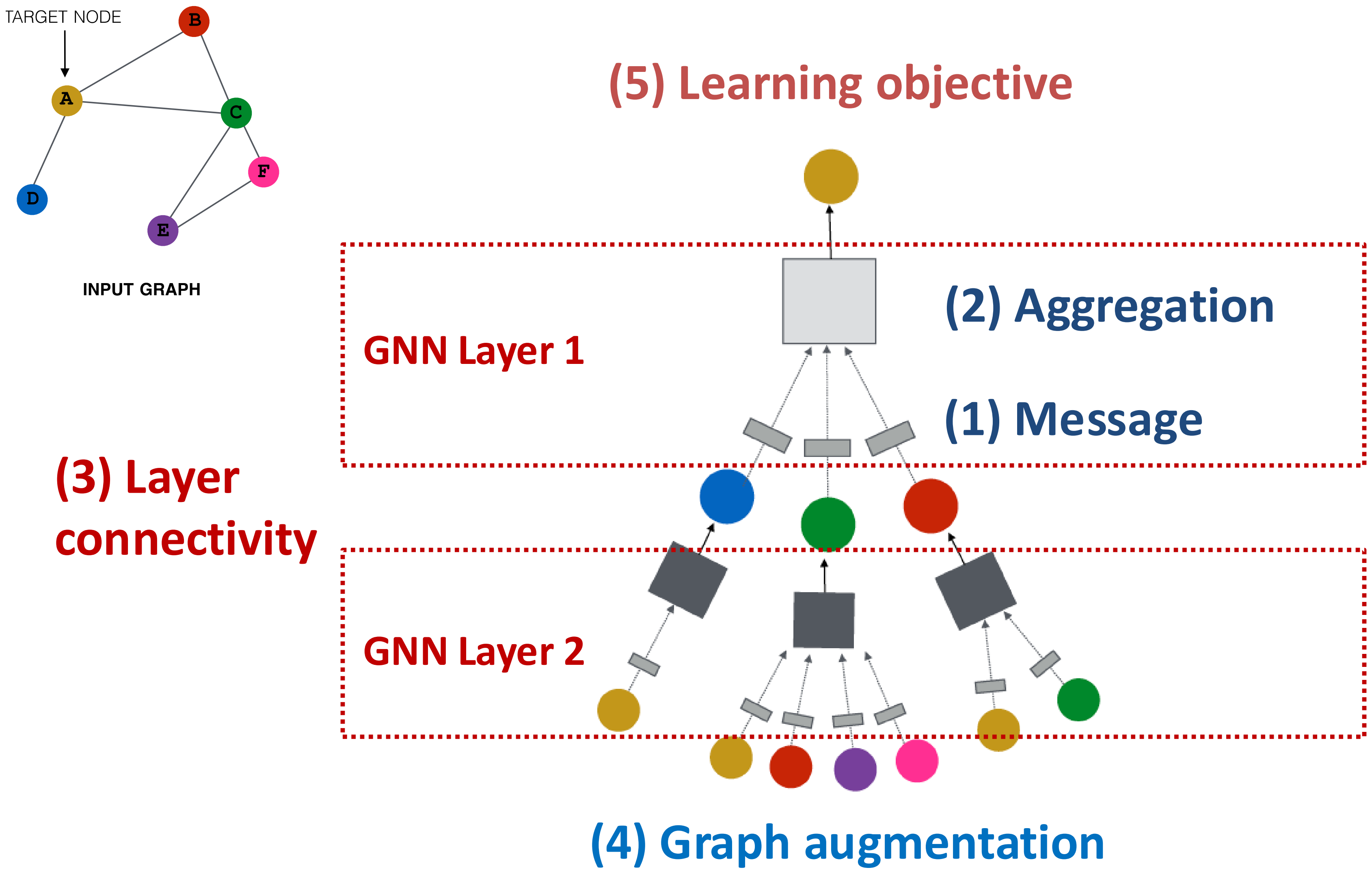


- Many application settings constantly encounter previously unseen nodes:
  - E.g., Reddit, YouTube, Google Scholar
- Need to generate new embeddings "on the fly"

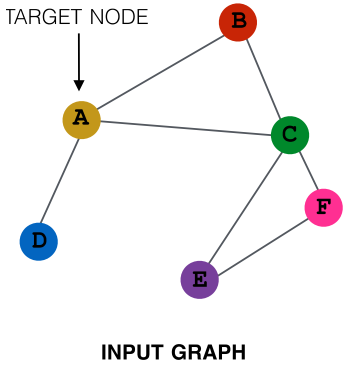
Graph Neural Networks

A General Perspective

A General GNN Framework

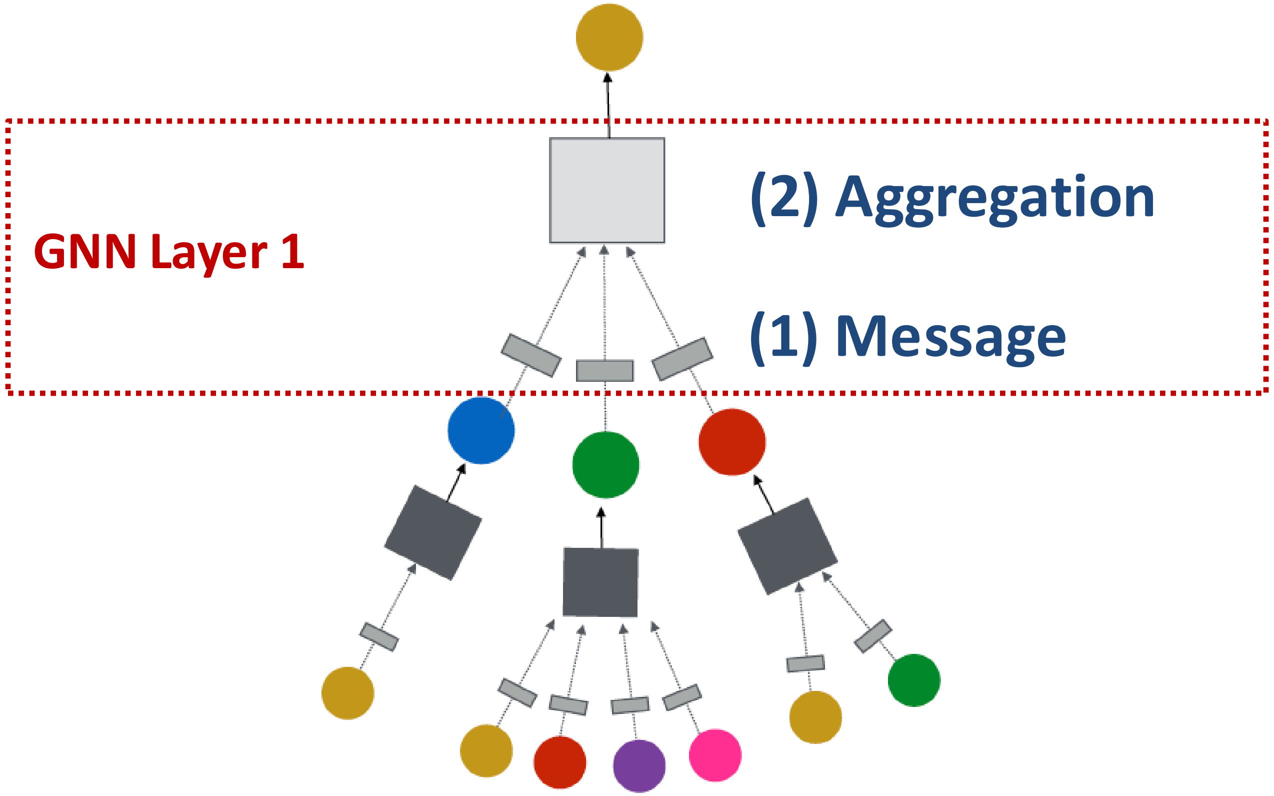


A General GNN Framework (1)

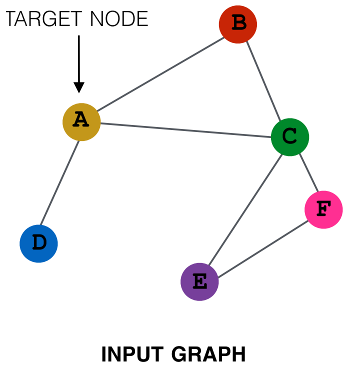


GNN Layer = Message + Aggregation

- **Different instantiations under this perspective**
- **GCN, GraphSAGE, GAT, …**

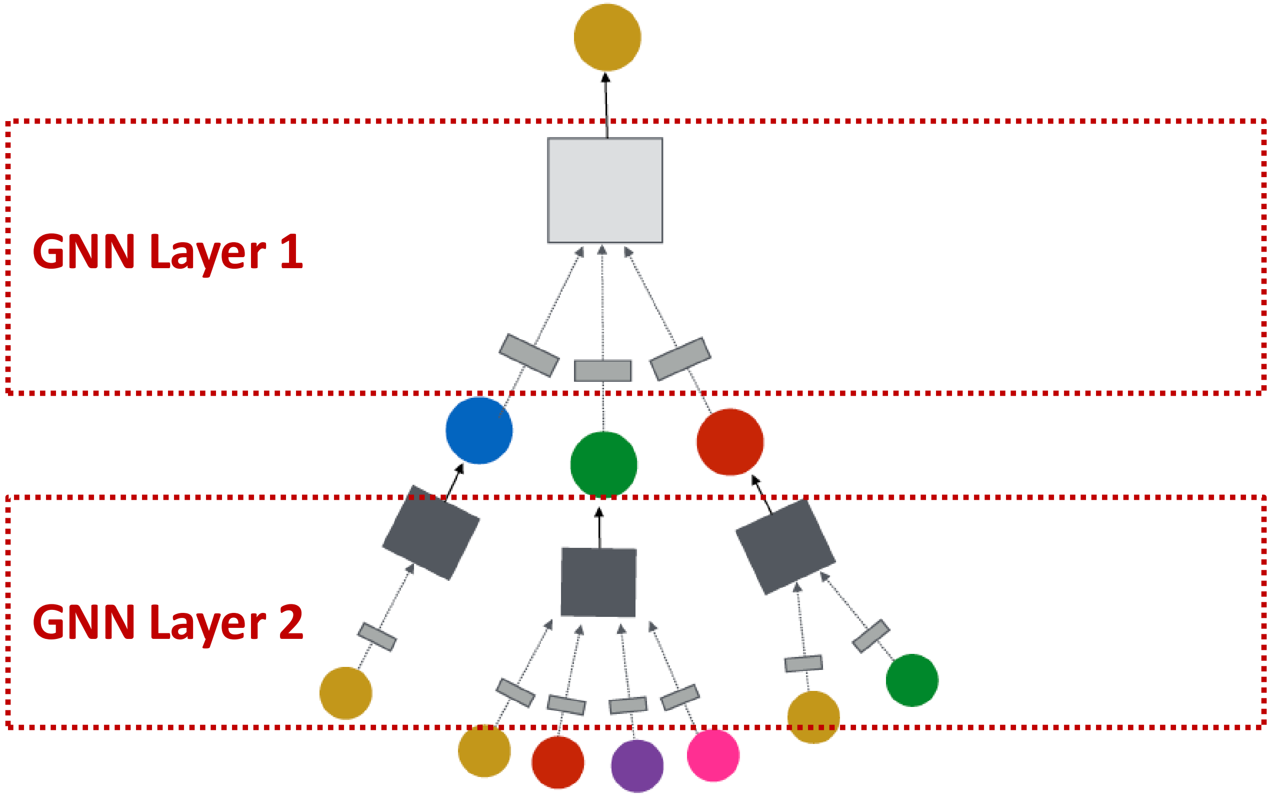


A General GNN Framework (2)



(3) Layer connectivity

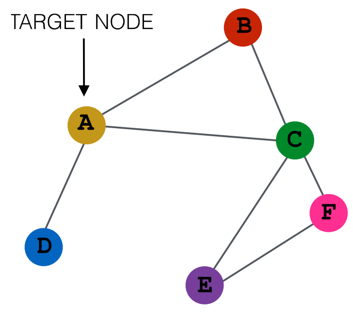
Connect GNN layers into a GNN
- **Stack layers sequentially**
- **Ways of adding skip connections**



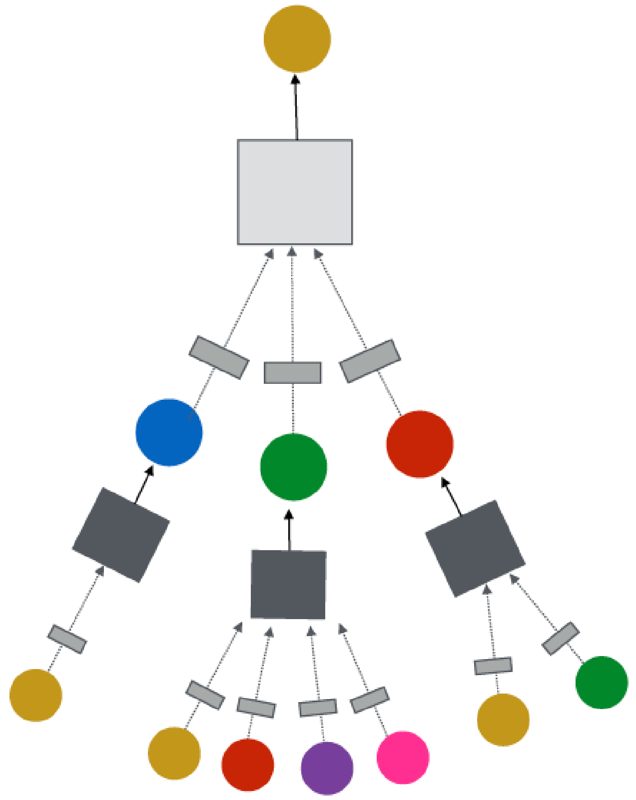
A General GNN Framework (3)

Idea: Raw input graph ≠ computational graph

- **Graph feature augmentation**
- **Graph structure augmentation**

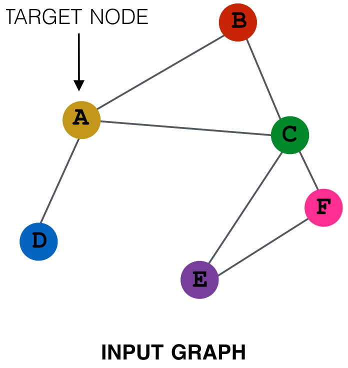


INPUT GRAPH

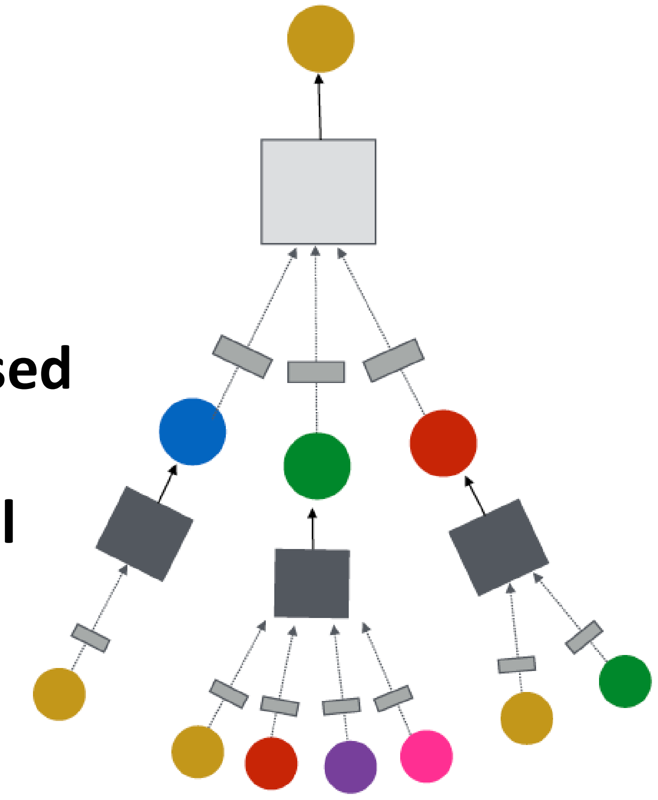


(4) Graph augmentation

A General GNN Framework (4)



(5) Learning objective

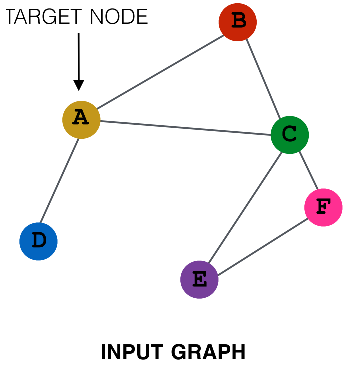


How do we train a GNN

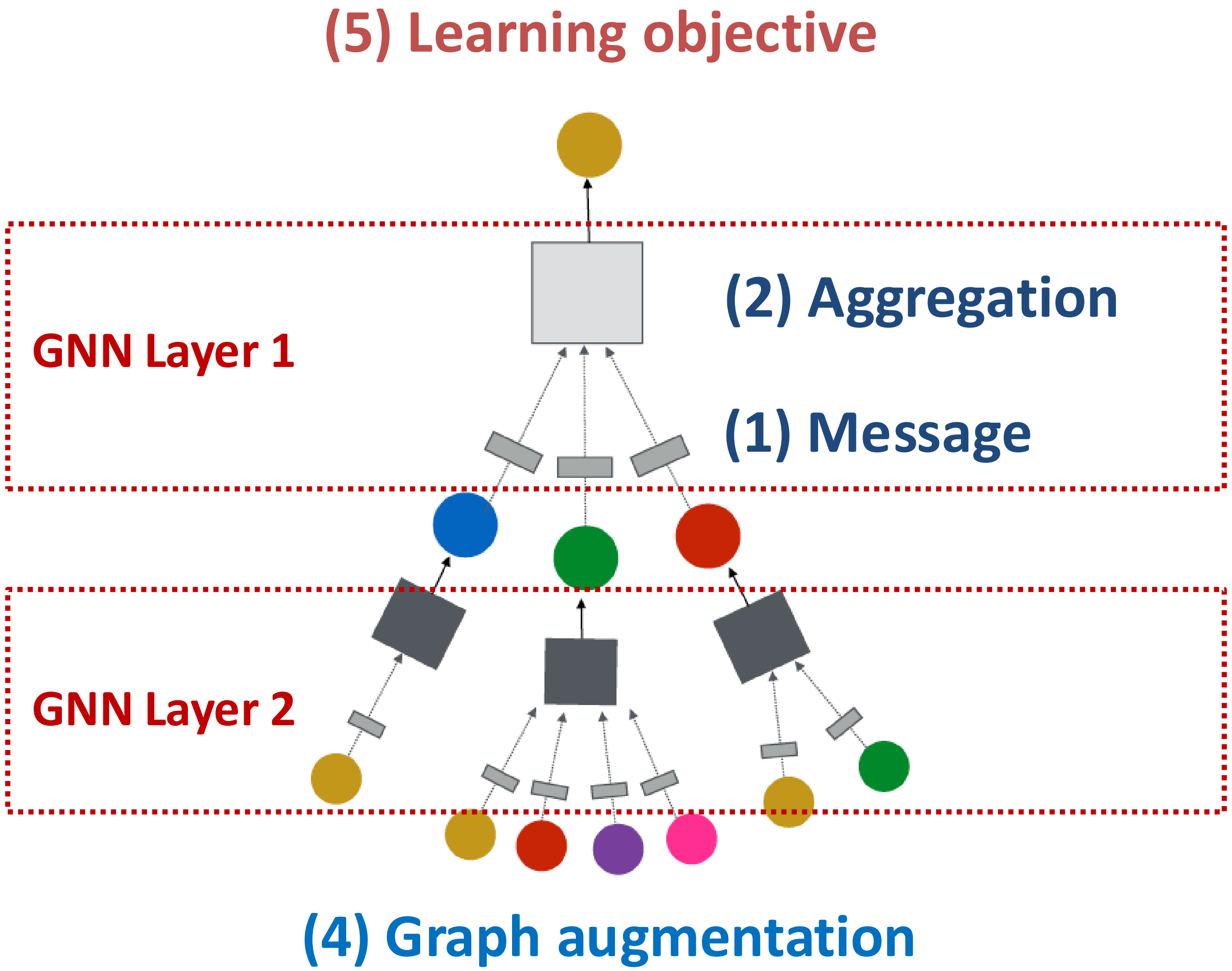
- **Supervised/Unsupervised objectives**
- **Node/Edge/Graph level objectives**

(We will discuss all of these later in class)

GNN Framework: Summary



Layer
nnectivity



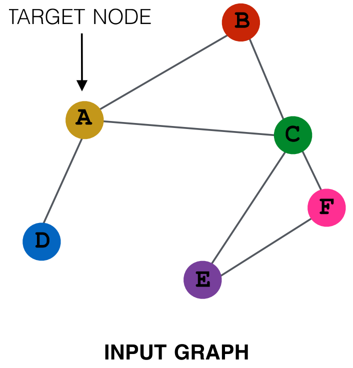
Graph Neural Networks: Model

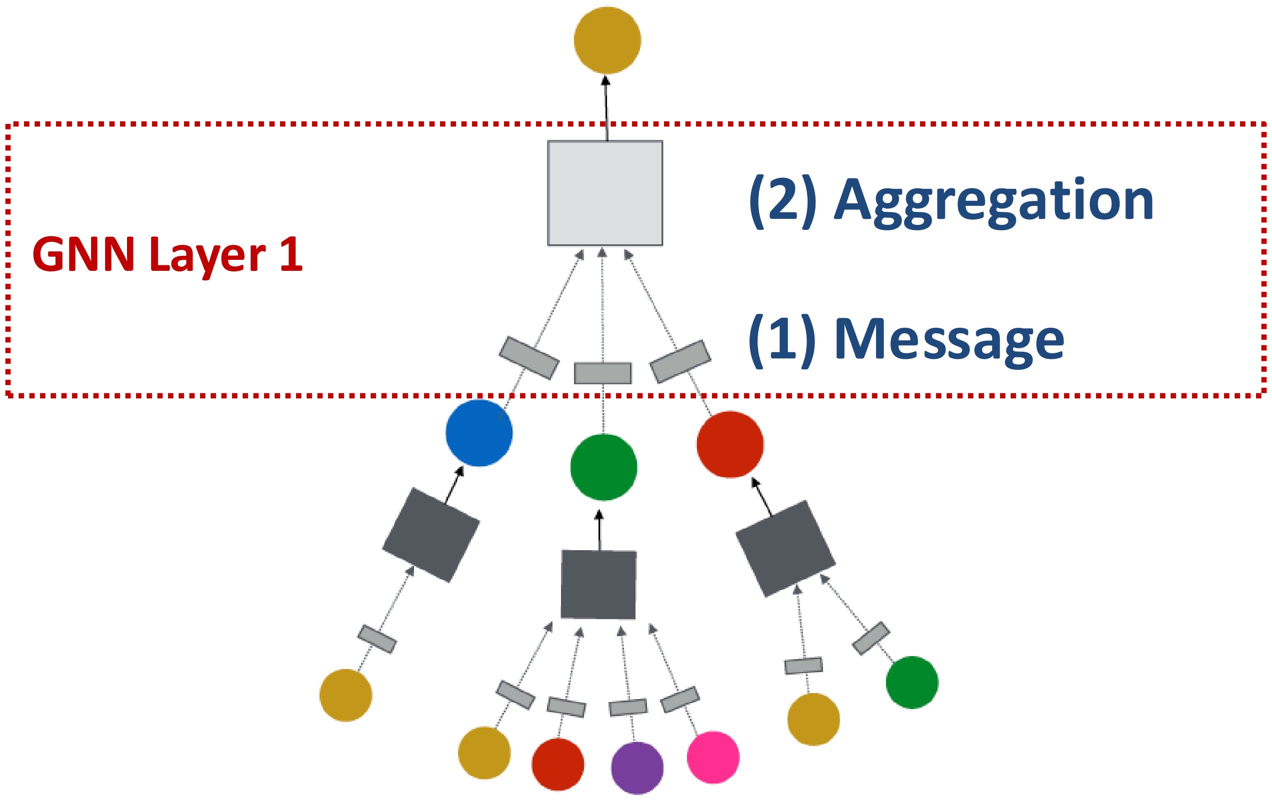
A Single Layer of a GNN

A GNN Layer

GNN Layer = Message + Aggregation

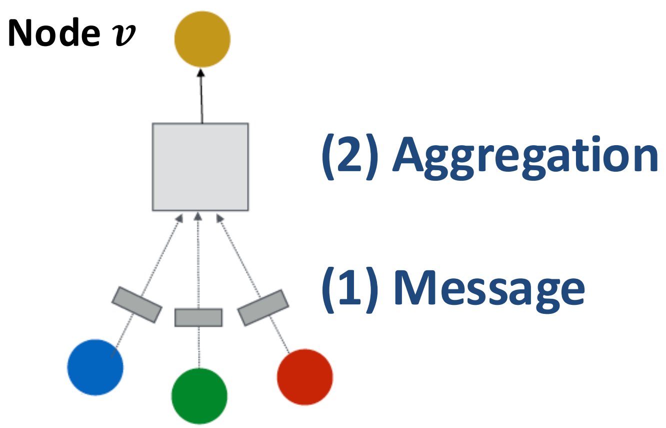
- **Different instantiations under this perspective**
- **GCN, GraphSAGE, GAT, …**

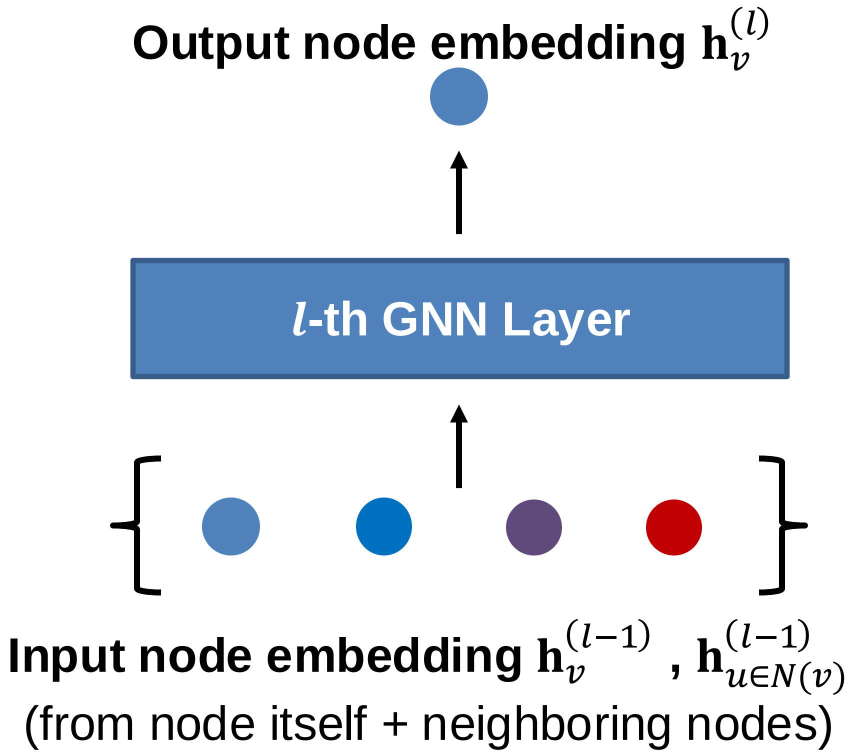




A Single GNN Layer

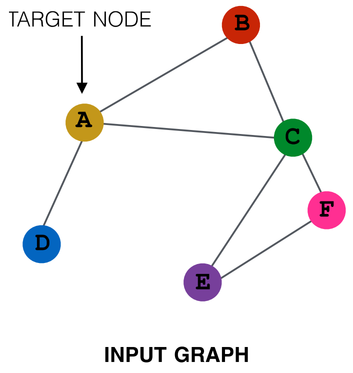
- **Idea of a GNN Layer:**
  - Compress a set of vectors into a single vector
  - **Two-step process:**
    - **(1) Message**
    - **(2) Aggregation**

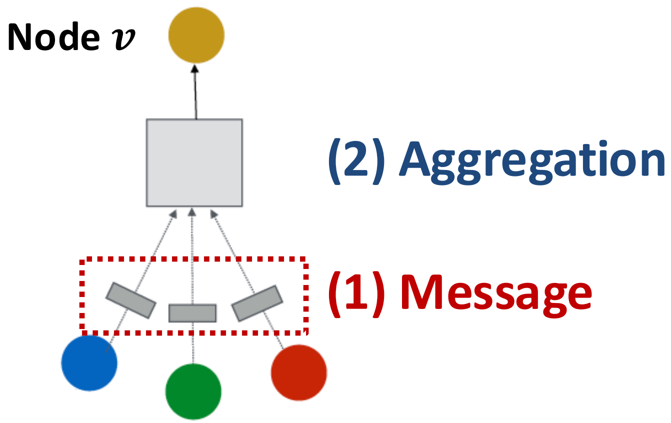




Message Computation

- **(1) Message computation**
  - **Message function:** $\mathbf{m}_u^{(l)} = \mathrm{MSG}^{(l)}\left(\mathbf{h}_u^{(l-1)}\right)$
    - **Intuition:** Each node will create a message, which will be sent to other nodes later
    - **Example:** A Linear layer $\mathbf{m}_u^{(l)} = \mathbf{W}^{(l)}\mathbf{h}_u^{(l-1)}$
      - Multiply node features with weight matrix $\mathbf{W}^{(l)}$



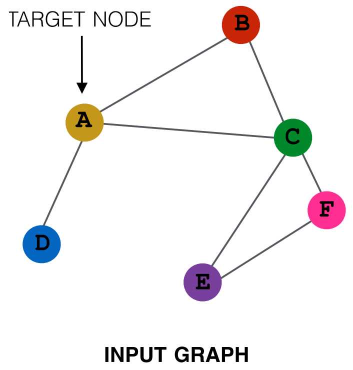


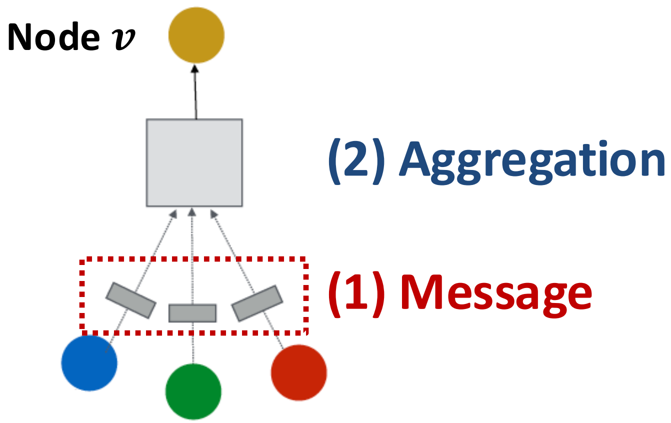
Message Aggregation

- **(2) Aggregation**
  - **Intuition:** Each node will aggregate the messages from node v's neighbors

$$\mathbf{h}_v^{(l)} = \mathrm{AGG}^{(l)}\left(\left\{\mathbf{m}_u^{(l)}, u \in N(v)\right\}\right)$$

  - **Example:** $\mathrm{Sum}(\cdot)$, $\mathrm{Mean}(\cdot)$ or $\mathrm{Max}(\cdot)$ aggregator
    - $\mathbf{h}_v^{(l)} = \mathrm{Sum}(\{\mathbf{m}_u^{(l)}, u \in N(v)\})$





Message Aggregation: Issue

- **Issue:** Information from node v itself **could get lost**
  - Computation of $\mathbf{h}_v^{(l)}$ does not directly depend on $\mathbf{h}_v^{(l-1)}$
- **Solution:** Include $\mathbf{h}_v^{(l-1)}$ when computing $\mathbf{h}_v^{(l)}$
  - **(1) Message: compute message from node v itself**
    - Usually, a **different message computation** will be performed

$$\mathbf{m}_u^{(l)} = \mathbf{W}^{(l)}\mathbf{h}_u^{(l-1)} \qquad \mathbf{m}_v^{(l)} = \mathbf{B}^{(l)}\mathbf{h}_v^{(l-1)}$$

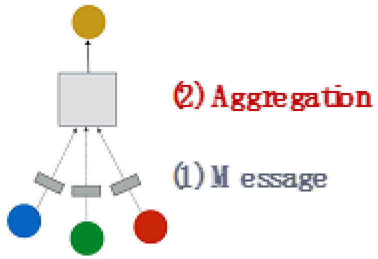
  - **(2) Aggregation:** After aggregating from neighbors, we can **aggregate the message from node v itself**
    - Via **concatenation** or **summation**

Then aggregate from node itself

$$\mathbf{h}_v^{(l)} = \mathrm{CONCAT}\left(\underbrace{\mathrm{AGG}\left(\left\{\mathbf{m}_u^{(l)}, u \in N(v)\right\}\right)}_{\text{First aggregate from neighbors}}, \boxed{\mathbf{m}_v^{(l)}}\right)$$

A Single GNN Layer

- **Putting things together:**
  - **(1) Message**: each node computes a message
    $$\mathbf{m}_u^{(l)} = \mathrm{MSG}^{(l)}\left(\mathbf{h}_u^{(l-1)}\right), u \in \{N(v) \cup v\}$$
  - **(2) Aggregation**: aggregate messages from neighbors
    $$\mathbf{h}_v^{(l)} = \mathrm{AGG}^{(l)}\left(\left\{\mathbf{m}_u^{(l)}, u \in N(v)\right\}, \mathbf{m}_v^{(l)}\right)$$
  - **Nonlinearity (activation):** Adds expressiveness
    - Often written as $\sigma(\cdot)$: $\mathrm{ReLU}(\cdot)$, $\mathrm{Sigmoid}(\cdot)$, …
    - Can be added to **message or aggregation**



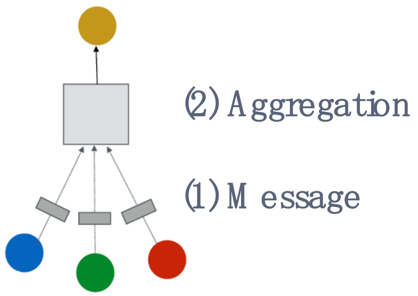
Classical GNN Layers: GCN (1)

- **(1) Graph Convolutional Networks (GCN)**

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

- **How to write this as Message + Aggregation?**

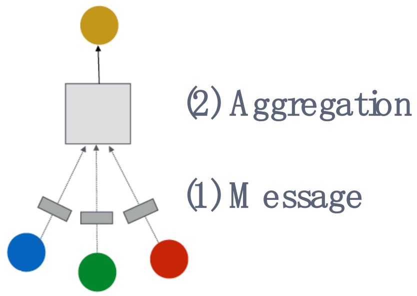
$$\mathbf{h}_v^{(l)} = \sigma\left(\underbrace{\sum_{u \in N(v)}}_{\textbf{Aggregation}} \overbrace{\mathbf{W}^{(l)} \frac{\mathbf{h}_u^{(l-1)}}{|N(v)|}}^{\textbf{Message}}\right)$$



Classical GNN Layers: GCN (2)

- **(1) Graph Convolutional Networks (GCN)**

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



- **Message:**
  - Each Neighbor: $\mathbf{m}_u^{(l)} = \frac{1}{|N(v)|}\mathbf{W}^{(l)}\mathbf{h}_u^{(l-1)}$

Normalized by node degree
(In the GCN paper they use a slightly different normalization)

- **Aggregation:**
  - **Sum** over messages from neighbors, then apply activation
  - $\mathbf{h}_v^{(l)} = \sigma\left(\mathrm{Sum}\left(\left\{\mathbf{m}_u^{(l)}, u \in N(v)\right\}\right)\right)$

In GCN graph is assumed to have self-edges that are included in the summation.

Classical GNN Layers: GraphSAGE

- **(2) GraphSAGE**

$$\mathbf{h}_v^{(l)} = \sigma\left(\mathbf{W}^{(l)} \cdot \mathrm{CONCAT}\left(\mathbf{h}_v^{(l-1)}, \mathrm{AGG}\left(\left\{\mathbf{h}_u^{(l-1)}, \forall u \in N(v)\right\}\right)\right)\right)$$

- **How to write this as Message + Aggregation?**
  - **Message** is computed within the $\mathrm{AGG}(\cdot)$
  - **Two-stage aggregation**
    - **Stage 1:** Aggregate from node neighbors

$$\mathbf{h}_{N(v)}^{(l)} \leftarrow \mathrm{AGG}\left(\left\{\mathbf{h}_u^{(l-1)}, \forall u \in N(v)\right\}\right)$$

    - **Stage 2:** Further aggregate over the node itself

$$\mathbf{h}_v^{(l)} \leftarrow \sigma\left(\mathbf{W}^{(l)} \cdot \mathrm{CONCAT}(\mathbf{h}_v^{(l-1)}, \mathbf{h}_{N(v)}^{(l)})\right)$$

GraphSAGE Neighbor Aggregation

- **Mean:** Take a weighted average of neighbors

$$\mathrm{AGG} = \underbrace{\boxed{\sum_{u \in N(v)}}}_{\textbf{Aggregation}} \underbrace{\frac{\mathbf{h}_u^{(l-1)}}{\boxed{|N(v)|}}}_{\textbf{Message computation}}$$

- **Pool:** Transform neighbor vectors and apply symmetric vector function $\mathrm{Mean}(\cdot)$ or $\mathrm{Max}(\cdot)$

$$\mathrm{AGG} = \underbrace{\boxed{\mathrm{Mean}}}_{\textbf{Aggregation}}(\{\underbrace{\boxed{\mathrm{MLP}}}_{\textbf{Message computation}}(\mathbf{h}_u^{(l-1)}), \forall u \in N(v)\})$$

- **LSTM:** Apply LSTM to reshuffled of neighbors

$$\mathrm{AGG} = \underbrace{\boxed{\mathrm{LSTM}}}_{\textbf{Aggregation}}([\mathbf{h}_u^{(l-1)}, \forall u \in \pi\big(N(v)\big)])$$

GraphSAGE: L2 Normalization

- ℓ_2 **Normalization:**
  - **Optional:** Apply ℓ_2 normalization to $\mathbf{h}_v^{(l)}$ at every layer
  - $\mathbf{h}_v^{(l)} \leftarrow \frac{\mathbf{h}_v^{(l)}}{\left\|\mathbf{h}_v^{(l)}\right\|_2} \; \forall v \in V$ where $\|u\|_2 = \sqrt{\sum_i u_i^2}$ (ℓ_2-norm)
  - Without ℓ_2 normalization, the embedding vectors have different scales (ℓ_2-norm) for vectors
  - In some cases (not always), normalization of embedding results in performance improvement
  - After ℓ_2 normalization, all vectors will have the same ℓ_2-norm

Classical GNN Layers: GAT (1)

- **(3) Graph Attention Networks**

$$\mathbf{h}_v^{(l)} = \sigma(\sum_{u \in N(v)} \underbrace{\boxed{\alpha_{vu}}}_{\text{Attention weights}} \mathbf{W}^{(l)} \mathbf{h}_u^{(l-1)})$$

- **In GCN / GraphSAGE**
  - $\alpha_{vu} = \frac{1}{|N(v)|}$ is the **weighting factor (importance)** of node u's message to node v
  - $\Longrightarrow \alpha_{vu}$ is defined **explicitly** based on the structural properties of the graph (node degree)
  - $\Longrightarrow$ All neighbors $u \in N(v)$ are equally important to node v

Classical GNN Layers: GAT (2)

- **(3) Graph Attention Networks**

$$\mathbf{h}_v^{(l)} = \sigma\left(\sum_{u \in N(v)} \boxed{\alpha_{vu}} \mathbf{W}^{(l)} \mathbf{h}_u^{(l-1)}\right)$$

Attention weights

Not all node's neighbors are equally important

- **Attention** is inspired by cognitive attention.
- The **attention** $\boldsymbol{\alpha_{vu}}$ focuses on the important parts of the input data and fades out the rest.
  - **Idea:** the NN should devote more computing power on that small but important part of the data.
  - Which part of the data is more important depends on the context and is learned through training.

Graph Attention Networks

Can we do better than simple neighborhood aggregation?

Can we let weighting factors α_{vu} to be learned?

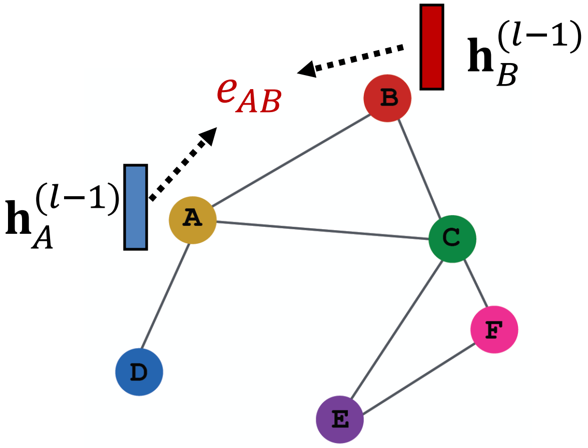
- **Goal:** Specify **arbitrary importance** to different neighbors of each node in the graph
- **Idea:** Compute embedding $\boldsymbol{h}_v^{(l)}$ of each node in the graph following an **attention strategy**:
  - Nodes attend over their neighborhoods' message
  - Implicitly specifying different weights to different nodes in a neighborhood

Attention Mechanism (1)

- Let α_{vu} be computed as a byproduct of an **attention mechanism $\boldsymbol{a}$:**
  - (1) Let a compute **attention coefficients $\boldsymbol{e_{vu}}$** across pairs of nodes u, v based on their messages:

$$e_{vu} = a(\mathbf{W}^{(l)}\mathbf{h}_u^{(l-1)}, \mathbf{W}^{(l)}\boldsymbol{h}_v^{(l-1)})$$

    - **$\boldsymbol{e_{vu}}$ indicates the importance of $\boldsymbol{u'}$s message to node $\boldsymbol{v}$ - logits**



$$e_{AB} = a(\mathbf{W}^{(l)}\mathbf{h}_A^{(l-1)}, \mathbf{W}^{(l)}\mathbf{h}_B^{(l-1)})$$

Attention Mechanism (2)

- **Normalize** e_{vu} into the **final attention weight** $\boldsymbol{\alpha_{vu}}$
  - Use the **softmax** function, so that $\sum_{u \in N(v)} \alpha_{vu} = 1$:

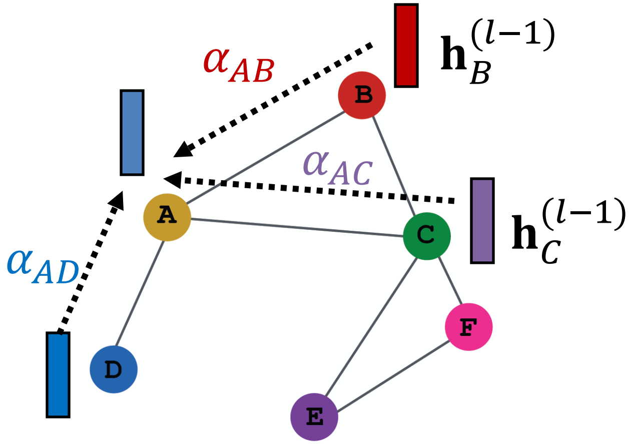
$$\alpha_{vu} = \frac{\exp(e_{vu})}{\sum_{k \in N(v)} \exp(e_{vk})}$$

- **Weighted sum** based on the **final attention weight** $\boldsymbol{\alpha_{vu}}$

$$\mathbf{h}_v^{(l)} = \sigma\left(\sum_{u \in N(v)} \alpha_{vu} \mathbf{W}^{(l)} \mathbf{h}_u^{(l-1)}\right)$$

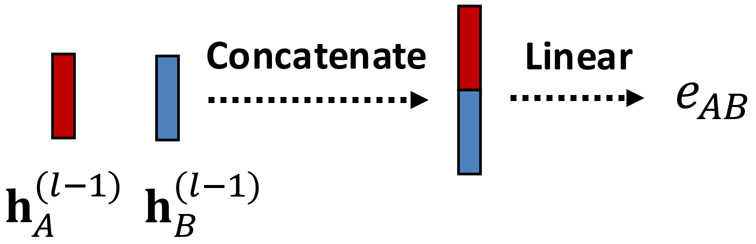
Weighted sum using α_{AB}, α_{AC}, α_{AD}:

$$\mathbf{h}_A^{(l)} = \sigma(\alpha_{AB}\mathbf{W}^{(l)}\mathbf{h}_B^{(l-1)} + \alpha_{AC}\mathbf{W}^{(l)}\mathbf{h}_C^{(l-1)} + \alpha_{AD}\mathbf{W}^{(l)}\mathbf{h}_D^{(l-1)})$$



Attention Mechanism (3)

- **What is the form of attention mechanism α?**
  - The approach is agnostic to the choice of α
    - E.g., use a simple single-layer neural network
      - α have trainable parameters (weights in the Linear layer)



$$e_{AB} = \alpha\left(\mathbf{W}^{(l)}\mathbf{h}_A^{(l-1)}, \mathbf{W}^{(l)}\mathbf{h}_B^{(l-1)}\right)$$
$$= \text{Linear}\left(\text{Concat}\left(\mathbf{W}^{(l)}\mathbf{h}_A^{(l-1)}, \mathbf{W}^{(l)}\mathbf{h}_B^{(l-1)}\right)\right)$$

  - Parameters of α are trained jointly:
    - Learn the parameters together with weight matrices (i.e., other parameter of the neural net $\mathbf{W}^{(l)}$) in an end-to-end fashion

Attention Mechanism (4)

- **Multi-head attention:** Stabilizes the learning process of attention mechanism
  - **Create multiple attention scores** (each replica with a different set of parameters):

$$\mathbf{h}_v^{(l)}[1] = \sigma(\textstyle\sum_{u \in N(v)} \alpha_{vu}^1 \mathbf{W}^{(l)} \mathbf{h}_u^{(l-1)})$$
$$\mathbf{h}_v^{(l)}[2] = \sigma(\textstyle\sum_{u \in N(v)} \alpha_{vu}^2 \mathbf{W}^{(l)} \mathbf{h}_u^{(l-1)})$$
$$\mathbf{h}_v^{(l)}[3] = \sigma(\textstyle\sum_{u \in N(v)} \alpha_{vu}^3 \mathbf{W}^{(l)} \mathbf{h}_u^{(l-1)})$$

  - **Outputs are aggregated:**
    - By concatenation or summation
    - $\mathbf{h}_v^{(l)} = \mathrm{AGG}(\mathbf{h}_v^{(l)}[1], \mathbf{h}_v^{(l)}[2], \mathbf{h}_v^{(l)}[3])$

Benefits of Attention Mechanism

- **Key benefit:** Allows for (implicitly) specifying **different importance values (α_{vu}) to different neighbors**
- **Computationally efficient**:
  - Computation of attentional coefficients can be parallelized across all edges of the graph
  - Aggregation may be parallelized across all nodes
- **Storage efficient**:
  - Sparse matrix operations do not require more than $O(V+E)$ entries to be stored
  - **Fixed** number of parameters, irrespective of graph size
- **Localized**:
  - Only **attends over local network neighborhoods**
- **Inductive capability**:
  - It is a shared *edge-wise* mechanism
  - It does not depend on the global graph structure

Summary of the lecture

- **GCN Pipeline**
- **A general perspective for GNNs**
  - **GNN Layer**:
    - Transformation + Aggregation
    - Classic GNN layers: GCN, GraphSAGE, GAT
- **Next:** GNN layer connectivity, graph manipulation