Graph Neural Networks: Model II

Jiaxuan You

Assistant Professor at UIUC CDS

CS598: Deep Learning with Graphs, 2024 Fall

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

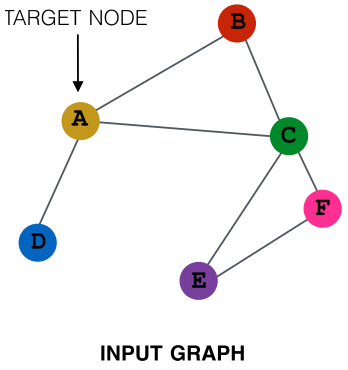
Logistics: Writing Task Due This Week

- All the responses should be submitted through **Canvas** in **comma-separated values (CSV)** files. 15% of grade.
- You can **download the CSV template** from a provided link.
- Please submit **one CSV file** for **each paper (3 total)**.
- Please indicate whether you are willing to share your input to the public.
  - It could be shared on our website to help other researchers gain insights, and we will acknowledge your name.
  - Don't worry, it's fully optional, choosing not to share your data will have no effect on your grades.
- **Submission DDL: Sept 15 (Sun) 11:59pm Central Time**

Logistics: Coding Homework

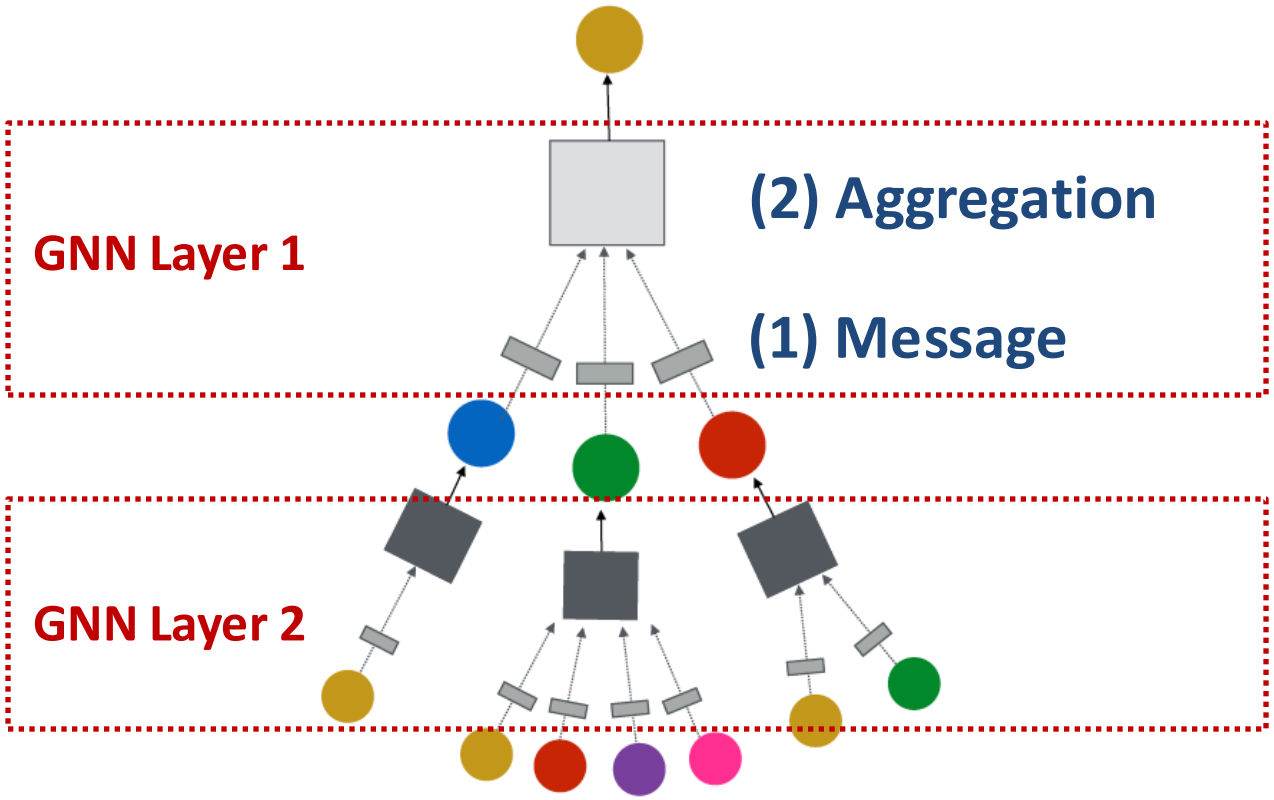
- Coding Assignment 1 Out
  - Assignment will be released on Canvas today.
  - Implement a full pipeline for learning node embeddings in Colab.
  - Submit your code and written answers downloaded from Colab to Canvas by **Sept 29 (Sun) 11:59 PM, CT.**

Recap: A General GNN Framework



(3) Layer connectivity

(5) Learning objective



(4) Graph augmentation

Recap: A Single GNN Layer

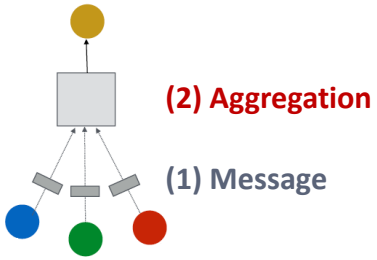
- **Putting things together:**
  - **(1) Message**: each node computes a message

$$\mathbf{m}_u^{(l)} = \mathrm{MSG}^{(l)}\left(\mathbf{h}_u^{(l-1)}\right), u \in \{N(v) \cup v\}$$

  - **(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**



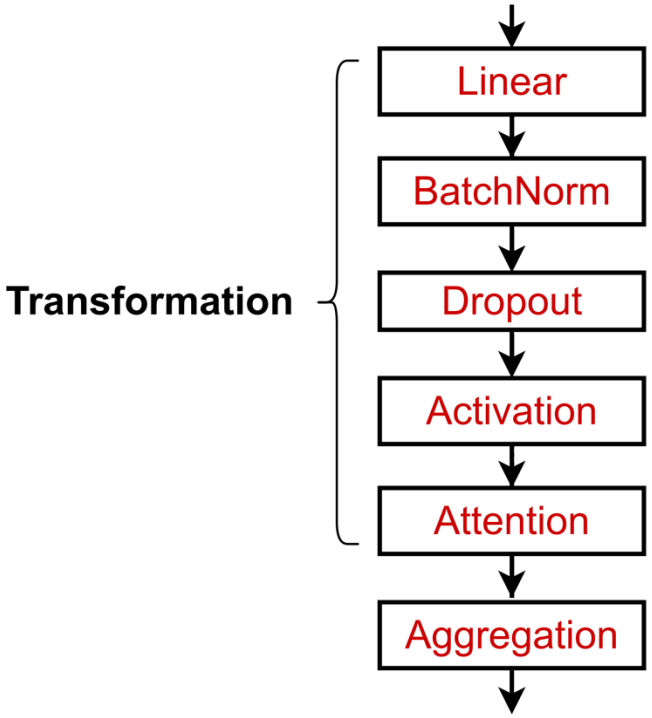
Graph Neural Networks: Model II

GNN Layers in Practice

GNN Layer in Practice

- **In practice, these classic GNN layers are a great starting point**
  - We can often get better performance by considerin a general GNN layer design
  - Concretely, we can include modern deep learning modules that proved to be useful in many domains

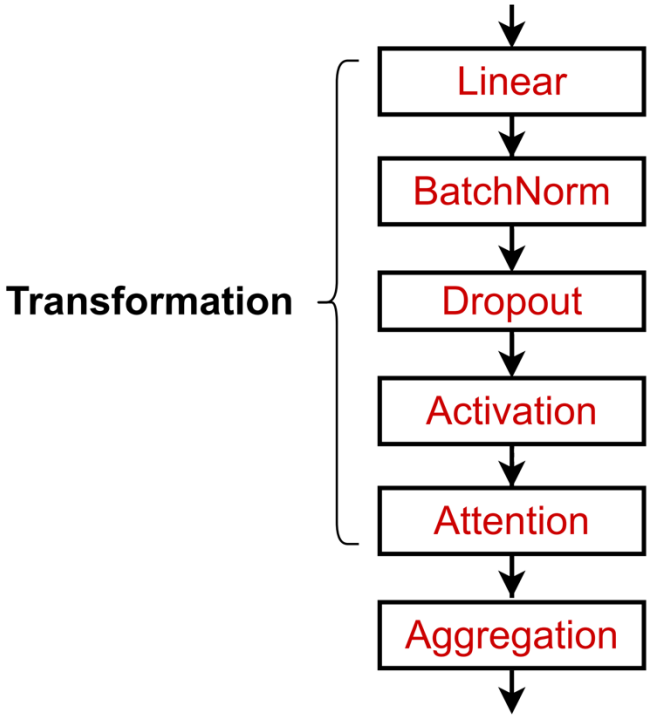
A suggested GNN Layer



GNN Layer in Practice

- **Many modern deep learning modules can be incorporated into a GNN layer**
  - **Batch Normalization:**
    - Stabilize neural network training
  - **Dropout:**
    - Prevent overfitting
  - **Attention/Gating:**
    - Control the importance of a message
  - **More:**
    - Any other useful deep learning modules

A suggested GNN Layer



Batch Normalization

- **Goal**: Stabilize neural networks training
- **Idea**: Given a batch of inputs (node embeddings)
  - Re-center the node embeddings into zero mean
  - Re-scale the variance into unit variance

Input: $\mathbf{X} \in \mathbb{R}^{N \times D}$
N node embeddings

Trainable Parameters:
$\boldsymbol{\gamma}, \boldsymbol{\beta} \in \mathbb{R}^{D}$

Output: $\mathbf{Y} \in \mathbb{R}^{N \times D}$
Normalized node embeddings

Step 1:
Compute the mean and variance over N embeddings

$$\boldsymbol{\mu}_j = \frac{1}{N}\sum_{i=1}^{N} \mathbf{X}_{i,j}$$

$$\boldsymbol{\sigma}_j^2 = \frac{1}{N}\sum_{i=1}^{N} \left(\mathbf{X}_{i,j} - \boldsymbol{\mu}_j\right)^2$$

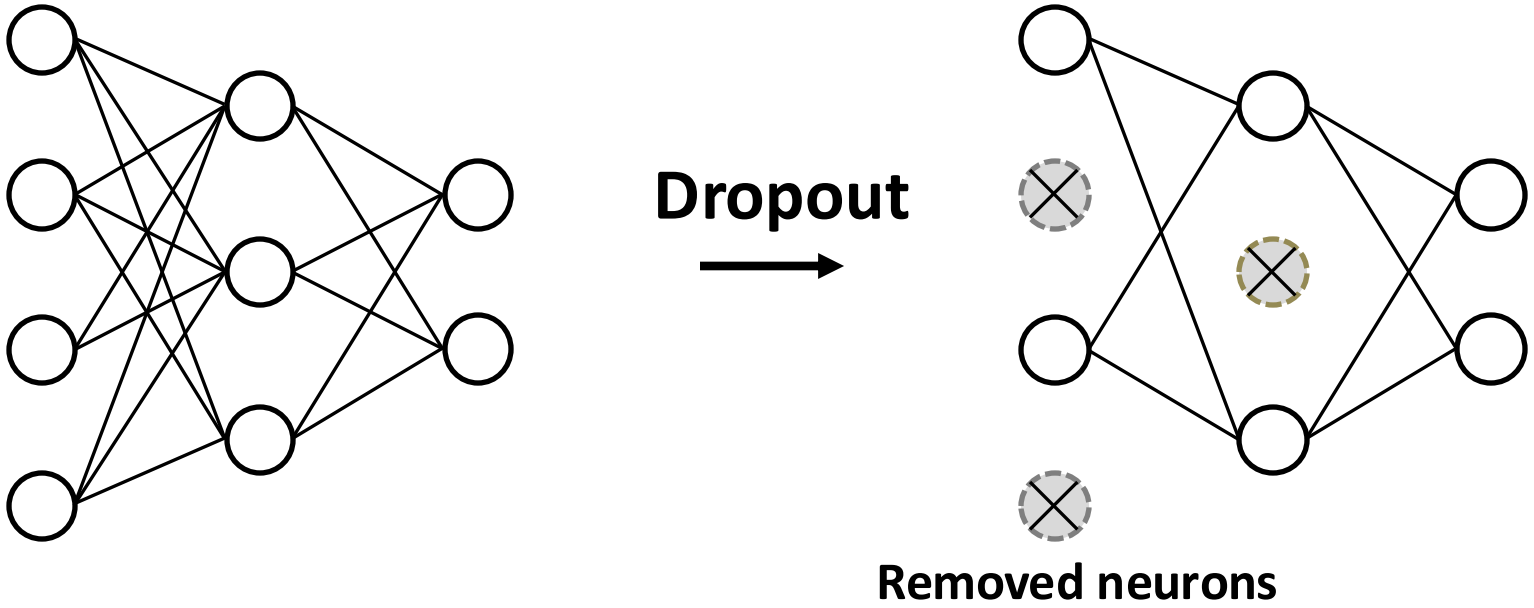
Step 2:
Normalize the feature using computed mean and variance

$$\widehat{\mathbf{X}}_{i,j} = \frac{\mathbf{X}_{i,j} - \boldsymbol{\mu}_j}{\sqrt{\boldsymbol{\sigma}_j^2 + \epsilon}}$$

$$\mathbf{Y}_{i,j} = \boldsymbol{\gamma}_j \widehat{\mathbf{X}}_{i,j} + \boldsymbol{\beta}_j$$

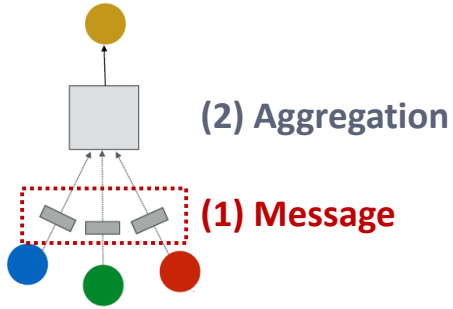
Dropout

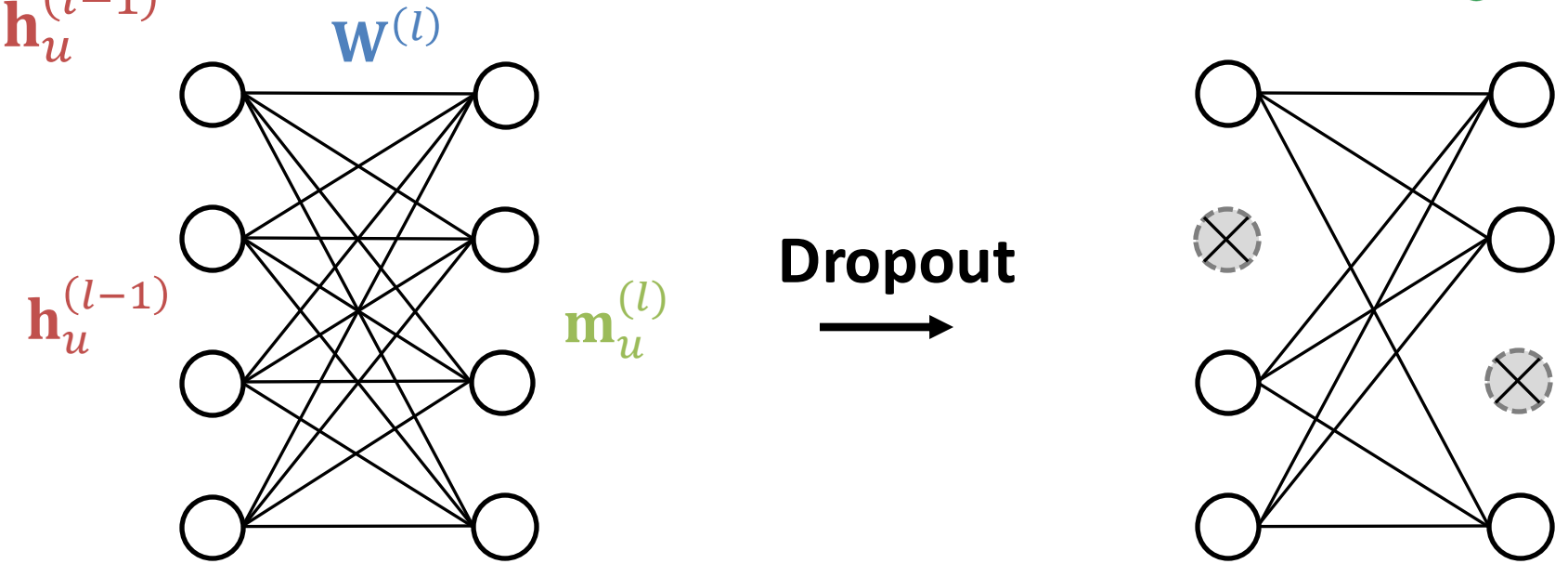
- **Goal**: Regularize a neural net to prevent overfitting.
- **Idea**:
  - **During training**: with some probability p, randomly set neurons to zero (turn off)
  - **During testing:** Use all the neurons for computation



Dropout for GNNs

- In GNN, Dropout is applied to **the <u>linear layer</u> in the message function**
  - **A simple message function with linear layer:** $\mathbf{m}_u^{(l)} = \mathbf{W}^{(l)}\mathbf{h}_u^{(l-1)}$





Visualization of a linear layer

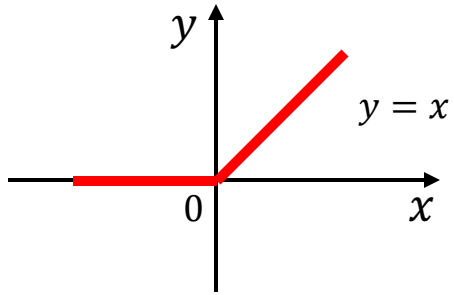
Activation (Non-linearity)

Apply activation to i-th dimension of embedding $\mathbf{x}$

- **Rectified linear unit (ReLU)**

  $$\mathrm{ReLU}(\mathbf{x}_i) = \max(\mathbf{x}_i, 0)$$

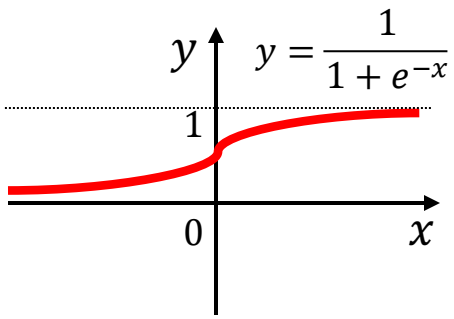
  - Most commonly used



- **Sigmoid**

  $$\sigma(\mathbf{x}_i) = \frac{1}{1 + e^{-\mathbf{x}_i}}$$

  - Used only when you want to restrict the range of your embeddings

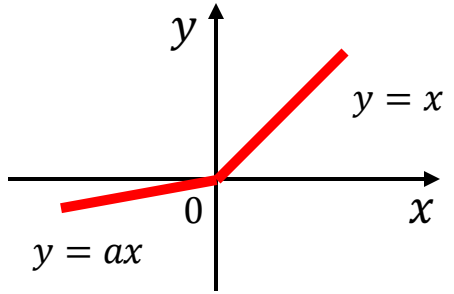


- **Parametric ReLU**

  $$\mathrm{PReLU}(\mathbf{x}_i) = \max(\mathbf{x}_i, 0) + a_i \min(\mathbf{x}_i, 0)$$

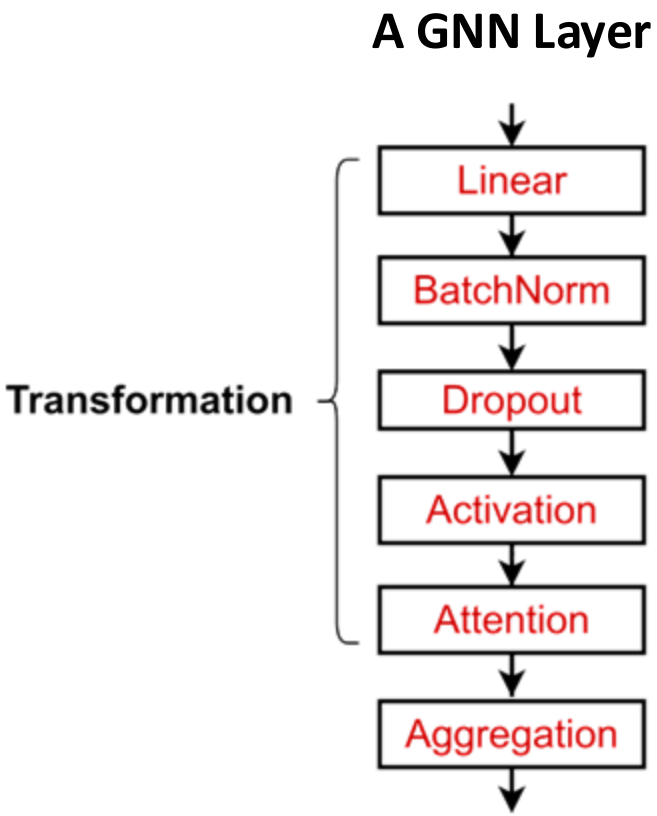
  a_i is a trainable parameter

  - Empirically performs better than ReLU



GNN Layer in Practice

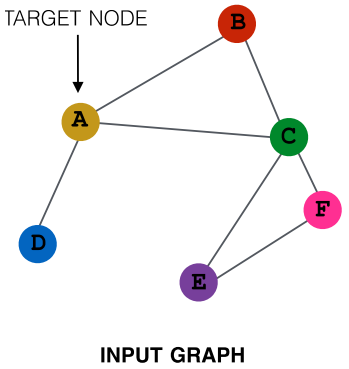
- **Summary:** Modern deep learning modules can be included into a GNN layer for better performance
- **Designing novel GNN layers is still an active research frontier!**
- **Suggested resources:** You can explore diverse GNN designs or try out your own ideas in **GraphGym**



Graph Neural Networks: Model II

Stacking Layers of a GNN

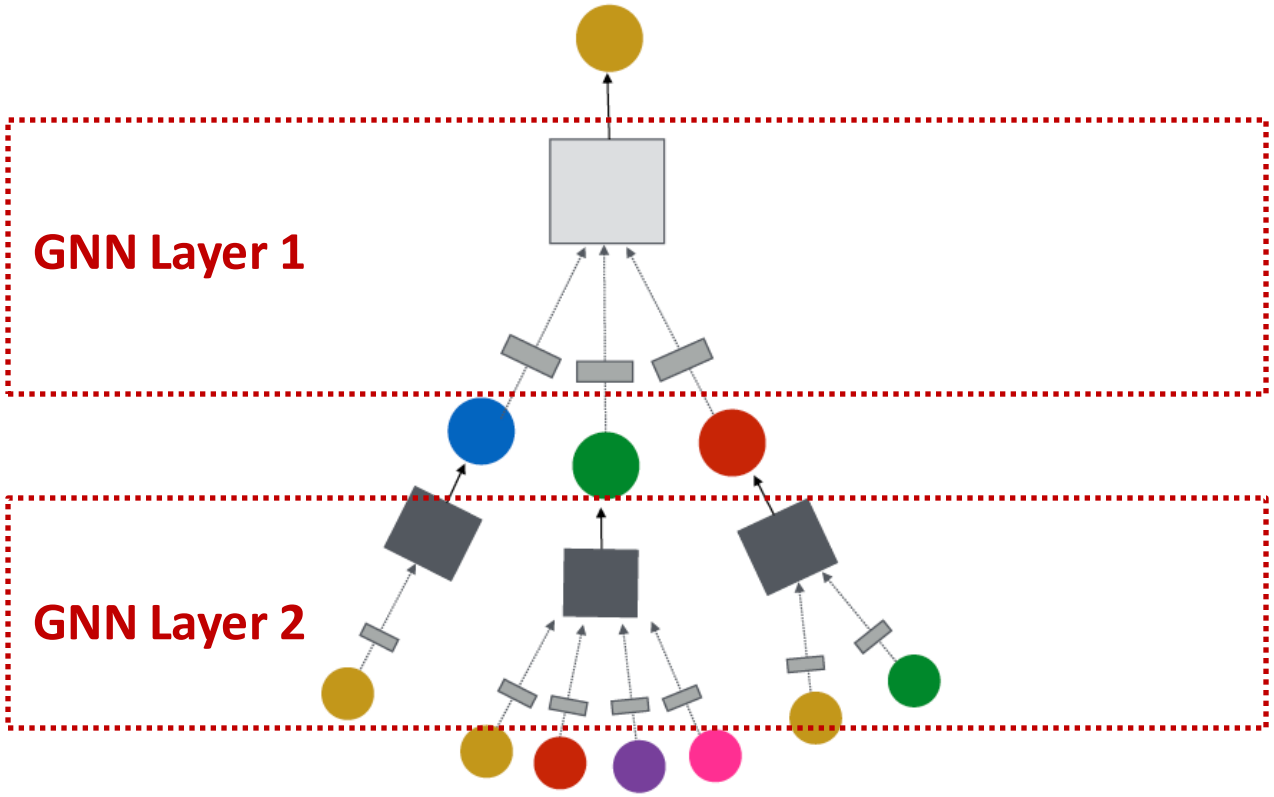
Stacking GNN Layers



How to connect GNN layers into a GNN?

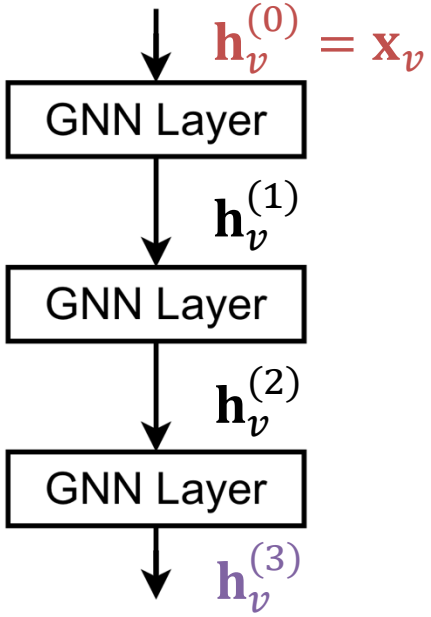
- **Stack layers sequentially**
- **Ways of adding skip connections**

(3) Layer connectivity



Stacking GNN Layers

- **How to construct a Graph Neural Network?**
  - **The standard way:** Stack GNN layers sequentially
  - **Input:** Initial raw node feature $\mathbf{x}_v$
  - **Output:** Node embeddings $\mathbf{h}_v^{(L)}$ after L GNN layers

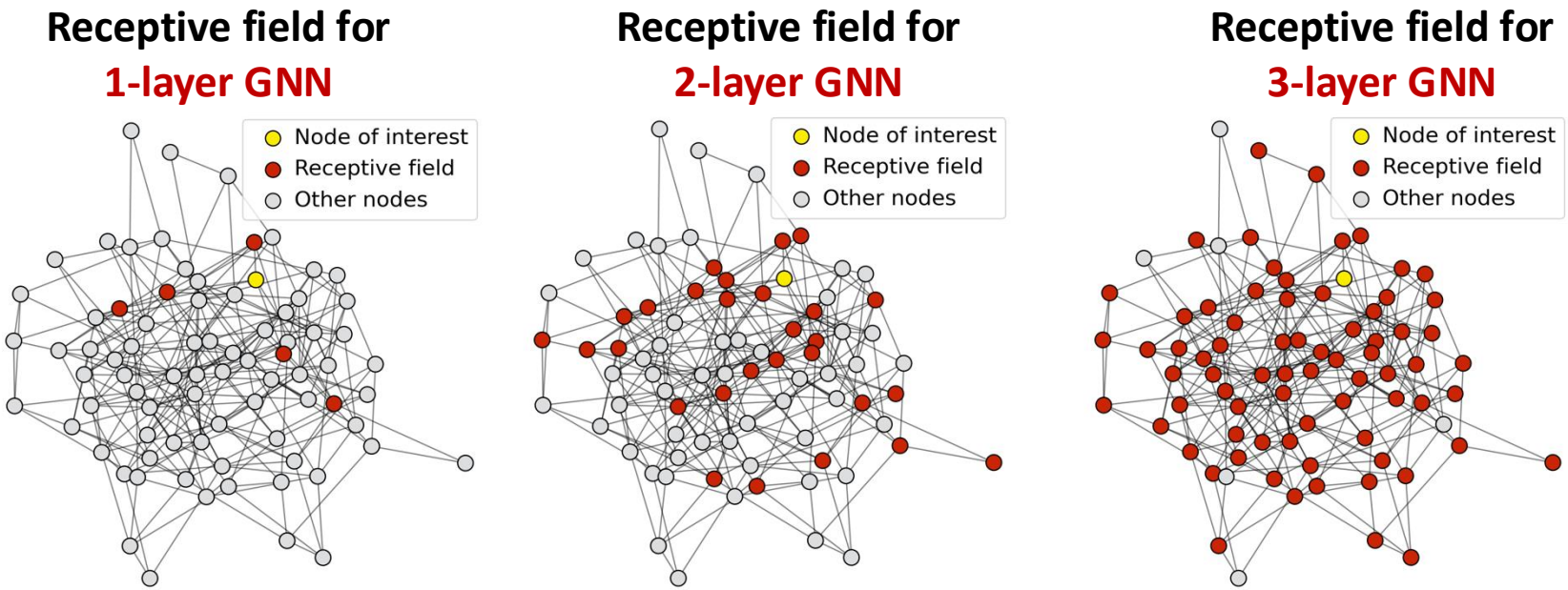


The Over-smoothing Problem

- **The Issue of stacking many GNN layers**
  - GNN suffers from **the over-smoothing problem**
- **The over-smoothing problem: all the node embeddings converge to the same value**
  - This is bad because we **want to use node embeddings to differentiate nodes**
- **Why does the over-smoothing problem happen?**

Receptive Field of a GNN

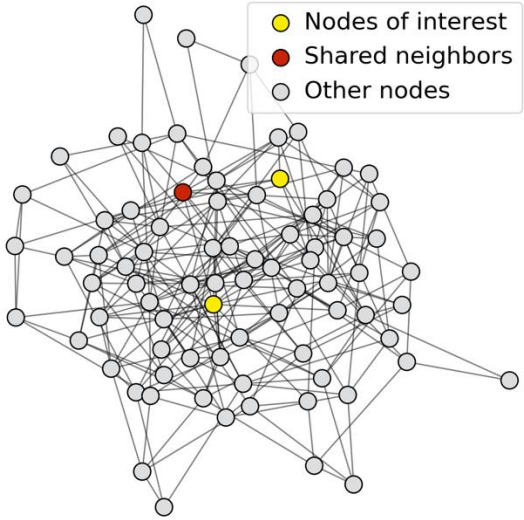
- **Receptive field:** the set of nodes that determine the embedding of a node of interest
  - **In a K-layer GNN, each node has a receptive field of K-hop neighborhood**



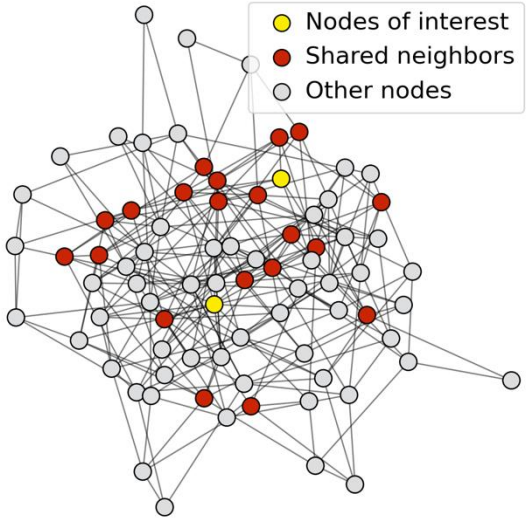
Receptive Field of a GNN

- **Receptive field overlap** for two nodes
  - **The shared neighbors quickly grows** when we increase the number of hops (num of GNN layers)

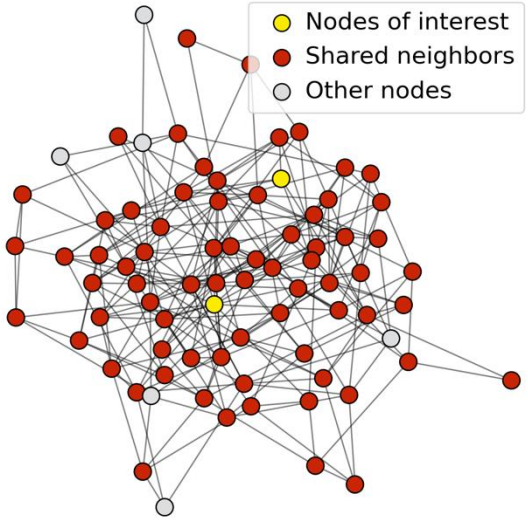
1-hop neighbor overlap
Only 1 node



2-hop neighbor overlap
About 20 nodes



3-hop neighbor overlap
Almost all the nodes!



Receptive Field & Over-smoothing

- **We can explain over-smoothing via the notion of receptive field**
  - We knew **the embedding of a node is determined by its receptive field**
    - If two nodes **have highly-overlapped receptive fields, then their embeddings are highly similar**
  - **Stack many GNN layers → nodes will have highly-overlapped receptive fields → node embeddings will be highly similar → suffer from the over-smoothing problem**
- **Next:** how do we overcome over-smoothing problem?

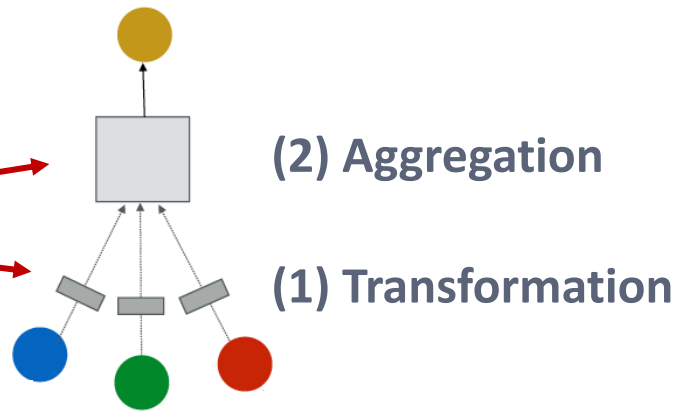
Design GNN Layer Connectivity

- **What do we learn from the over-smoothing problem?**
- **Lesson 1: Be cautious when adding GNN layers**
  - Unlike neural networks in other domains (CNN for image classification), **adding more GNN layers do not always help**
  - **Step 1: Analyze the necessary receptive field** to solve your problem. E.g., by computing the diameter of the graph
  - **Step 2:** Set number of GNN layers L to be a bit more than the receptive field we like. **Do not set L to be unnecessarily large**!
- **Question:** How to enhance the expressive power of a GNN, if the number of GNN layers is small?

Expressive Power for Shallow GNNs

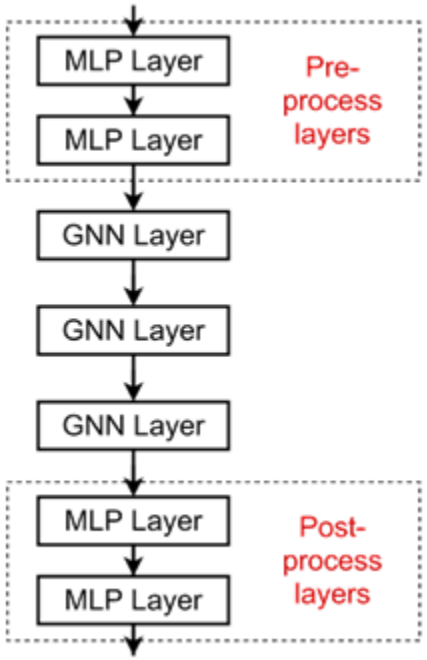
- **How to make a shallow GNN more expressive?**
- **Solution 1:** Increase the expressive power **within each GNN layer**
  - In our previous examples, each transformation or aggregation function only include one linear layer
  - We can **make aggregation / transformation become a deep neural network**!

If needed, each box could include a 3-layer MLP



Expressive Power for Shallow GNNs

- **How to make a shallow GNN more expressive?**
- **Solution 2:** Add layers that do not pass messages
  - A GNN does not necessarily only contain GNN layers
    - E.g., we can add **MLP layers** (applied to each node) before and after GNN layers, as **pre-process layers** and **post-process layers**



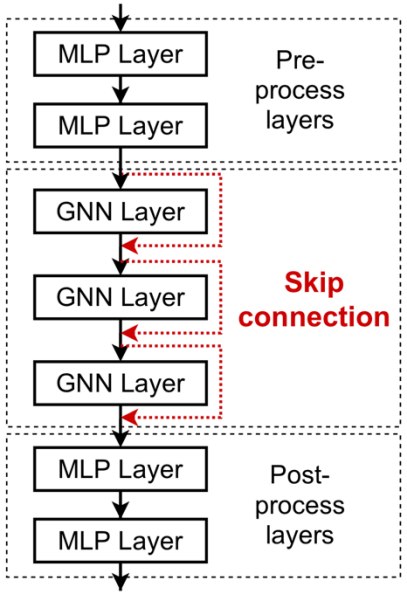
Pre-processing layers: Important when encoding node features is necessary.
E.g., when nodes represent images/text

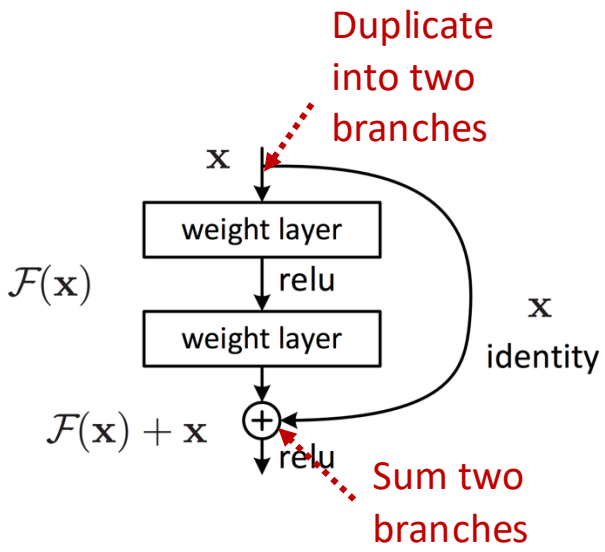
Post-processing layers: Important when reasoning / transformation over node embeddings are needed
E.g., graph classification, knowledge graphs

In practice, adding these layers works great!

Design GNN Layer Connectivity

- **What if my problem still requires many GNN layers?**
- **Lesson 2: Add skip connections in GNNs**
  - **Observation from over-smoothing:** Node embeddings in earlier GNN layers can sometimes better differentiate nodes
  - **Solution:** We can increase the impact of earlier layers on the final node embeddings, **by adding shortcuts in GNN**





Idea of skip connections:

Before adding shortcuts:

$$\boldsymbol{F}(\mathbf{x})$$

After adding shortcuts:

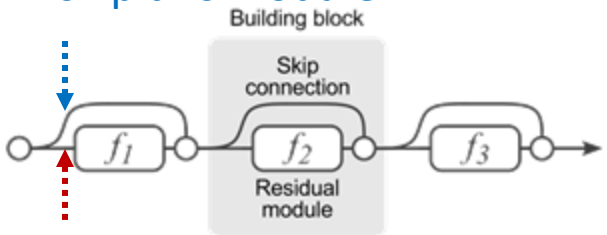
$$\boldsymbol{F}(\mathbf{x}) + \mathbf{x}$$

Veit et al. Residual Networks Behave Like Ensembles of Relatively Shallow Networks, ArXiv 2016

Idea of Skip Connections

- **Why do skip connections work?**
  - **Intuition:** Skip connections create **a mixture of models**
  - N skip connections → 2^N possible paths
  - Each path could have up to N modules
- We automatically get **a mixture of shallow GNNs and deep GNNs**

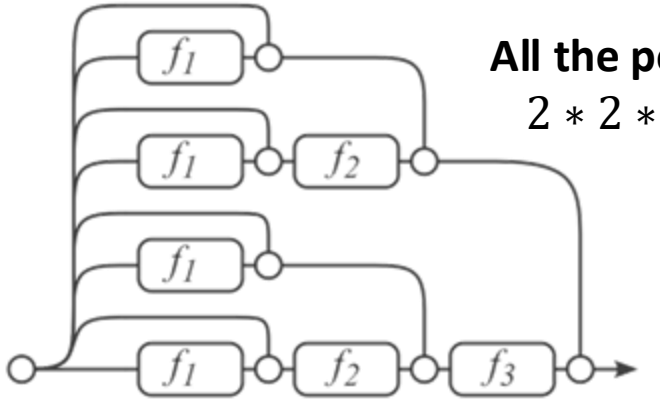
Path 2: skip this module



Path 1: include this module

(a) Conventional 3-block residual network

=



(b) Unraveled view of (a)

All the possible paths:
$2 * 2 * 2 = 2^3 = 8$

Example: GCN with Skip Connections

- **A standard GCN layer**

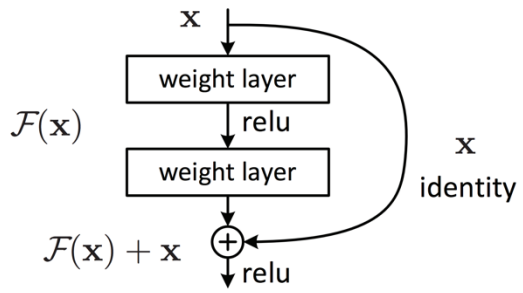
- $\mathbf{h}_v^{(l)} = \sigma\left(\boxed{\sum_{u \in N(v)} \mathbf{W}^{(l)} \frac{\mathbf{h}_u^{(l-1)}}{|N(v)|}}\right)$

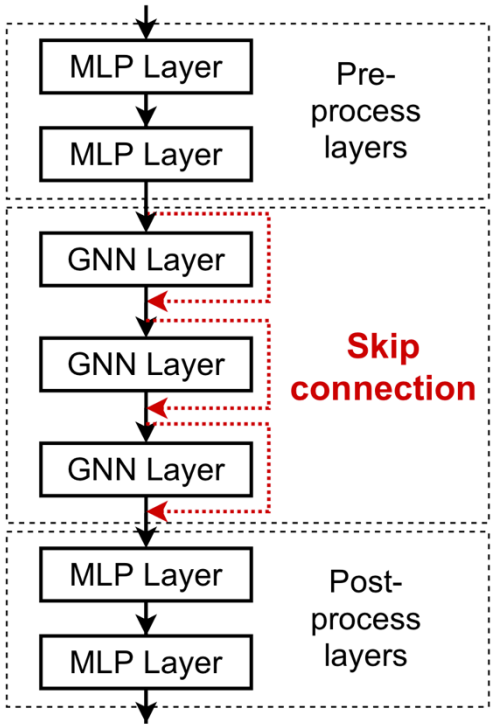
  This is our $F(\mathbf{x})$

- **A GCN layer with skip connection**

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

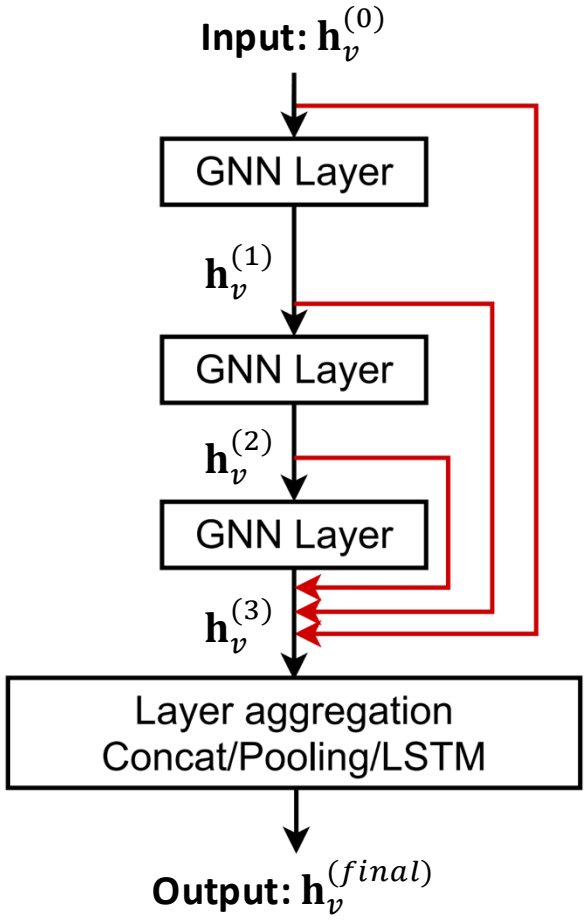
  $F(\mathbf{x}) \quad + \quad \mathbf{x}$





Other Options of Skip Connections

- **Other options:** Directly skip to the last layer
  - The final layer directly **aggregates from the all the node embeddings** in the previous layers



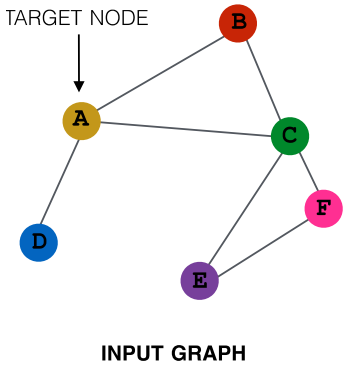
Graph Neural Networks: Model II

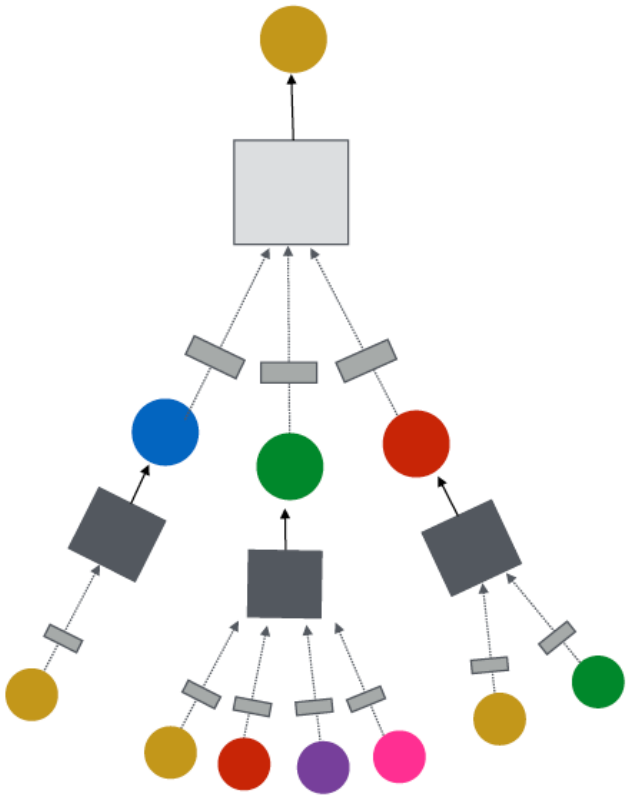
GNN Augmentation and Training

General GNN Framework

Idea: Raw input graph ≠ computational graph

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





(4) Graph manipulation

Why Manipulate Graphs

Our assumption so far has been

- **Raw input graph = computational graph**

Reasons for breaking this assumption

- **Feature level:**
  - The input graph **lacks features** → feature augmentation
- **Structure level:**
  - The graph is **too sparse** → inefficient message passing
  - The graph is **too dense** → message passing is too costly
  - The graph is **too large** → cannot fit the computational graph into a GPU
- It's just **unlikely that the input graph happens to be the optimal computation graph** for embeddings

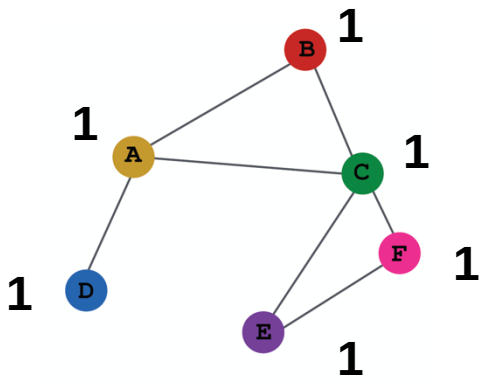
Graph Manipulation Approaches

- **Graph Feature manipulation**
  - The input graph **lacks features** → **feature augmentation**
- **Graph Structure manipulation**
  - The graph is **too sparse** → **Add virtual nodes / edges**
  - The graph is **too dense** → **Sample neighbors when doing message passing**
  - The graph is **too large** → **Sample subgraphs to compute embeddings**
    - Will cover later in lecture: Scaling up GNNs

Feature Augmentation on Graphs

Why do we need feature augmentation?

- **(1) Input graph does not have node features**
  - This is common when we only have the adj. matrix
- **Standard approaches:**
- **a) Assign constant values to nodes**

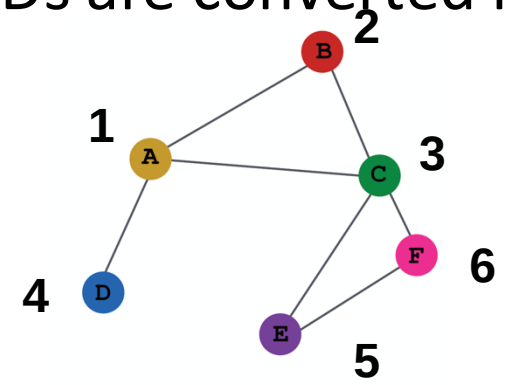


INPUT GRAPH

Feature Augmentation on Graphs

Why do we need feature augmentation?

- **(1) Input graph does not have node features**
  - This is common when we only have the adj. matrix
- **Standard approaches:**
- **b) Assign unique IDs to nodes**
  - These IDs are converted into **one-hot vectors**



INPUT GRAPH

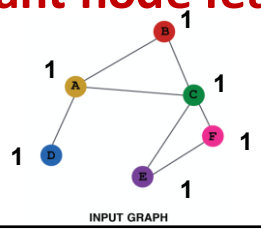
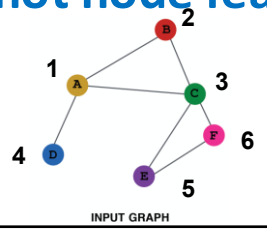
One-hot vector for node with ID=5

ID = 5

$$\underbrace{[0, 0, 0, 0, 1, 0]}_{\text{Total number of IDs = 6}}$$

Feature Augmentation on Graphs

- Feature augmentation: **constant** vs. **one-hot**

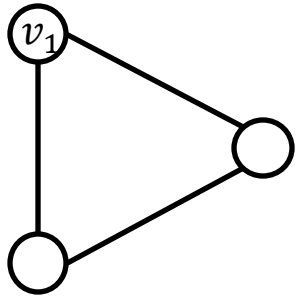
	Constant node feature	**One-hot node feature**
		
Expressive power	**Medium**. All the nodes are identical, but GNN can still learn from the graph structure	**High**. Each node has a unique ID, so node-specific information can be stored
Inductive learning (Generalize to unseen nodes)	**High**. Simple to generalize to new nodes: we assign constant feature to them, then apply our GNN	**Low**. Cannot generalize to new nodes: new nodes introduce new IDs, GNN doesn't know how to embed unseen IDs
Computational cost	**Low**. Only 1 dimensional feature	**High**. High dimensional feature, cannot apply to large graphs
Use cases	Any graph, inductive settings (generalize to new nodes)	Small graph, transductive settings (no new nodes)

Feature Augmentation on Graphs

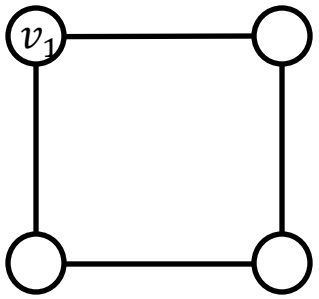
Why do we need feature augmentation?

- **(2) Certain structures are hard to learn by GNN**
- **Example:** Cycle count feature
  - Can GNN learn the length of a cycle that v_1 resides in?
  - **Unfortunately, no**

v_1 resides in a cycle with length 3



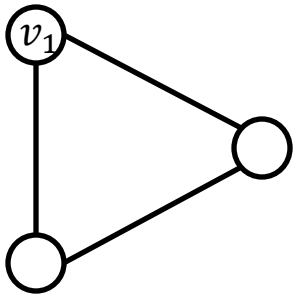
v_1 resides in a cycle with length 4



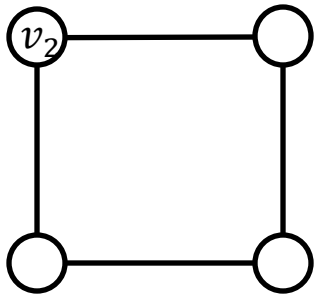
Feature Augmentation on Graphs

- v_1 **cannot differentiate which graph it resides in**
  - Because all the nodes in the graph have degree of 2
  - The computational graphs will be the same binary tree

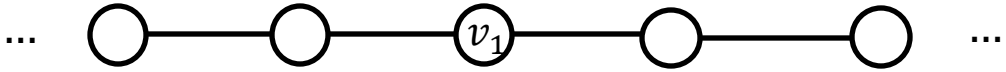
v_1 resides in a cycle with length 3



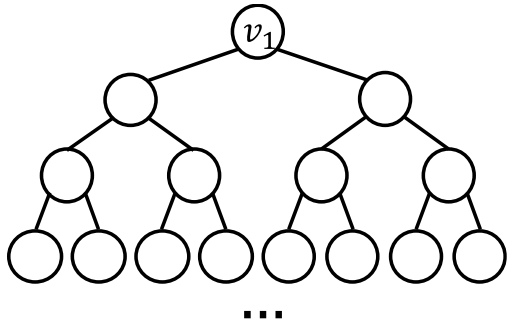
v_1 resides in a cycle with length 4



v_1 resides in a cycle with infinite length



The computational graphs for node v_1 are always the same



Feature Augmentation on Graphs

Why do we need feature augmentation?

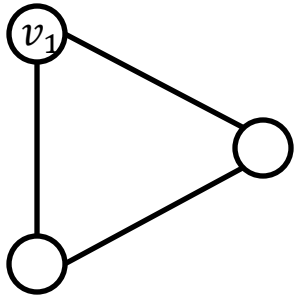
- **(2) Certain structures are hard to learn by GNN**
- **Solution:**
  - We can use cycle count as augmented node features

We start from cycle with length 0

Augmented node feature for v_1

[0, 0, 0, 1, 0, 0]

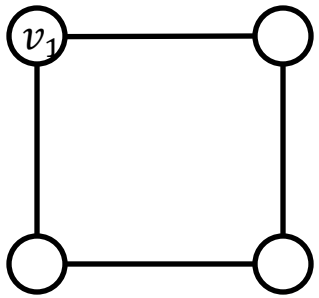
v_1 resides in a cycle with length 3



Augmented node feature for v_1

[0, 0, 0, 0, 1, 0]

v_1 resides in a cycle with length 4



Feature Augmentation on Graphs

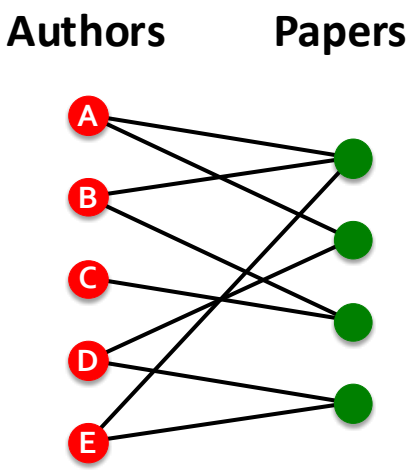
Why do we need feature augmentation?

- **(2) Certain structures are hard to learn by GNN**
- Other commonly used augmented features:
  - **Degree distribution**
  - **Clustering coefficient**
  - **PageRank**
  - **Centrality**
  - **…**
- Any **feature we have introduced can be used!**

Add Virtual Nodes / Edges

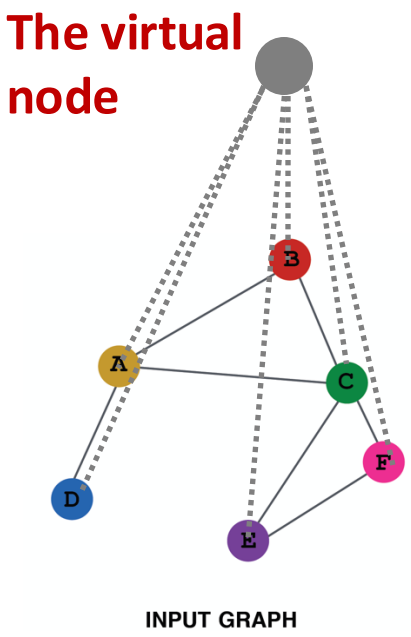
- **Motivation:** Augment sparse graphs
- **(1) Add virtual edges**
  - **Common approach:** Connect 2-hop neighbors via virtual edges
  - **Intuition:** Instead of using adj. matrix A for GNN computation, use $A + A^2$

- **Use cases:** Bipartite graphs
  - Author-to-papers (they authored)
  - 2-hop virtual edges make an author-author collaboration graph



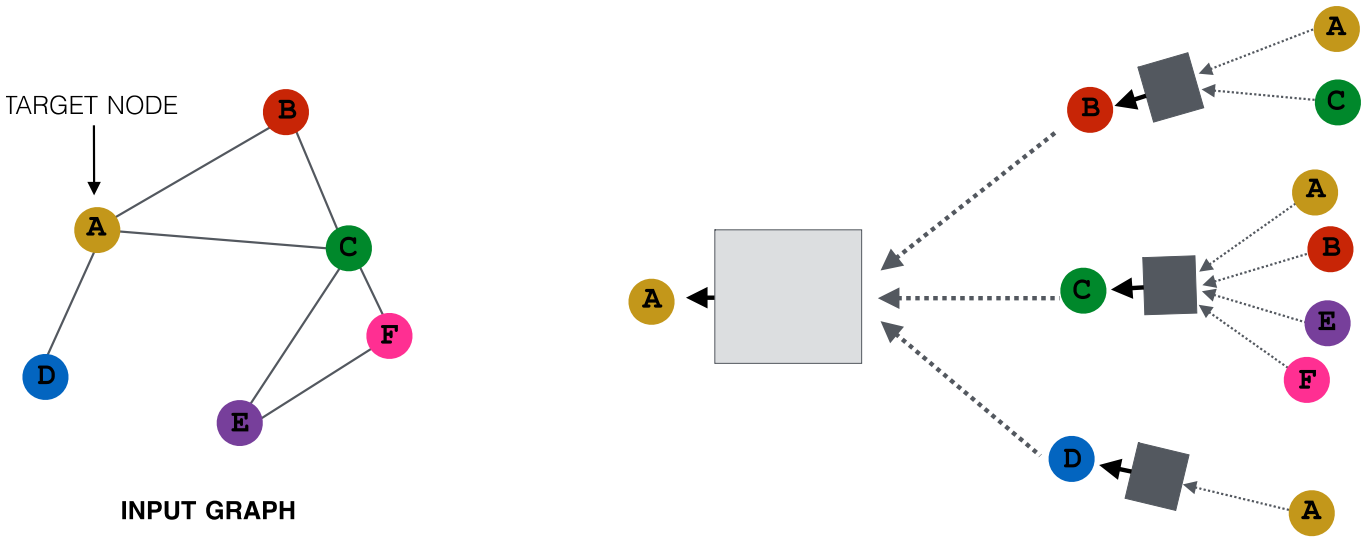
Add Virtual Nodes / Edges

- **Motivation:** Augment sparse graphs
- **(2) Add virtual nodes**
  - The virtual node will connect to all the nodes in the graph
    - Suppose in a sparse graph, two nodes have shortest path distance of 10
    - After adding the virtual node, **all the nodes will have a distance of 2**
      - **Node A – Virtual node – Node B**
  - **Benefits:** Greatly **improves message passing in sparse graphs**



Node Neighborhood Sampling

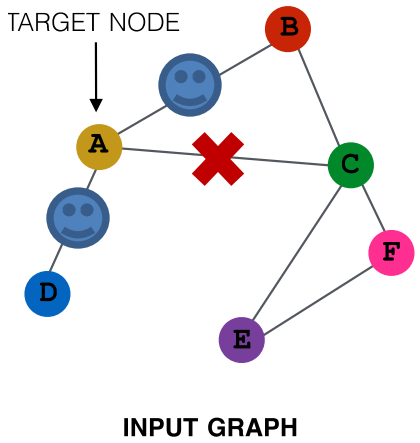
- **Previously:**
  - All the nodes are used for message passing

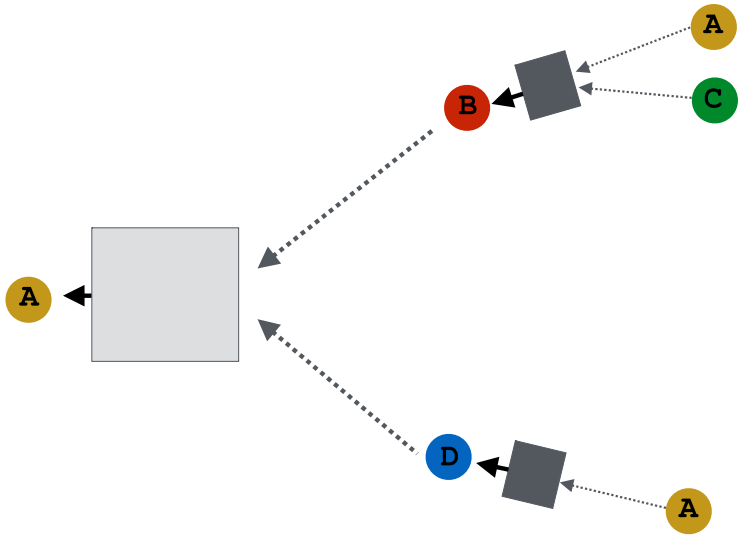


- **New idea:** (Randomly) sample a node's neighborhood for message passing

Neighborhood Sampling Example

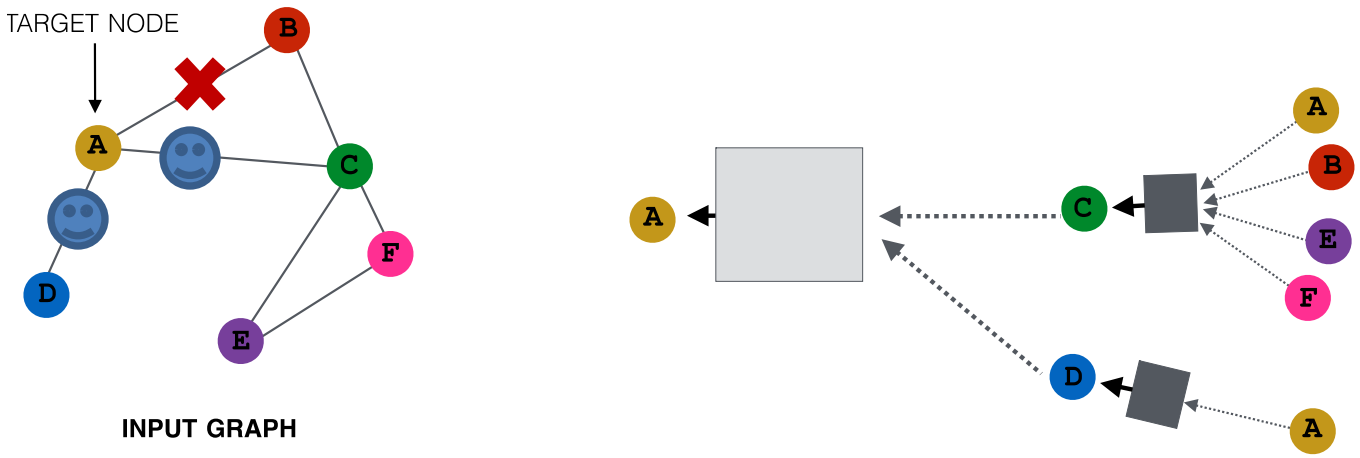
- **For example, we can randomly choose 2 neighbors to pass messages**
  - Only nodes B and D will pass message to A





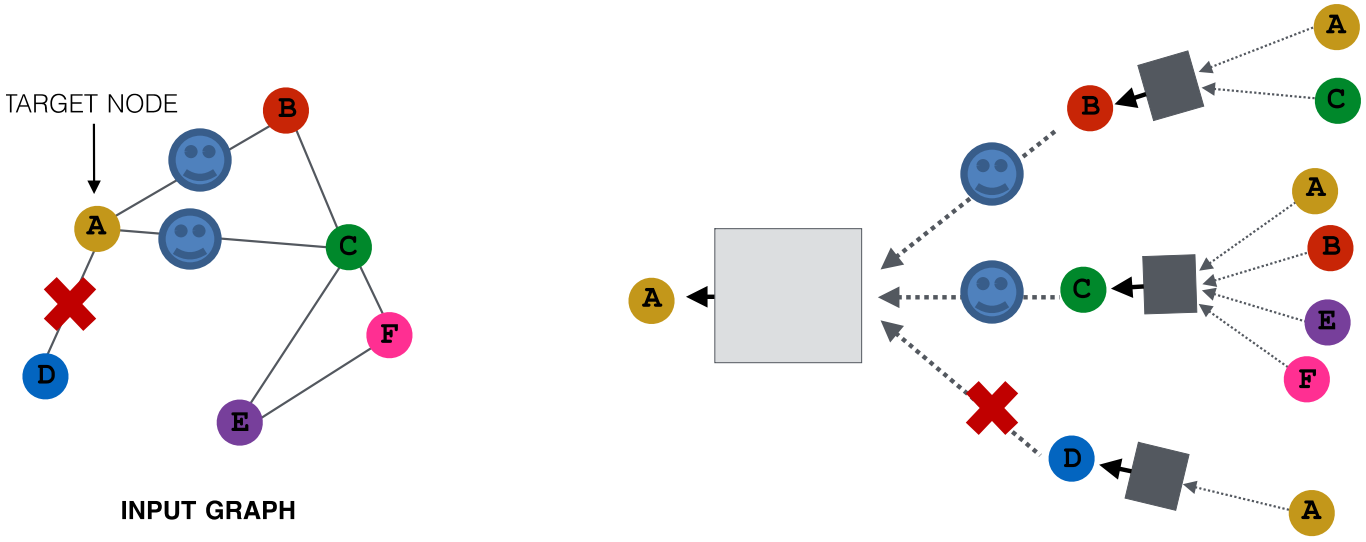
Neighborhood Sampling Example

- **Next time when we compute the embeddings, we can sample different neighbors**
  - Only nodes C and D will pass message to A



Neighborhood Sampling Example

- In expectation, we can get embeddings similar to the case where all the neighbors are used
  - **Benefits:** Greatly reduce computational cost
  - And in practice it works great!



Summary of the Lecture

- **We introduce a general GNN framework:**
  - **GNN Layer**:
    - Transformation + Aggregation
    - Classic GNN layers: GCN, GraphSAGE, GAT
  - **Layer connectivity**:
    - The over-smoothing problem
    - Solution: skip connections
  - **Graph Augmentation:**
    - Feature augmentation
    - Structure augmentation

Paper Discussion & Teammate Finding

Discussion Format & Objectives

Discussion Format & Objectives

- Our discussions and brainstorm will center on the 5 questions:
  - What is the problem?
  - Why is it interesting and important?
  - Why is it hard? (E.g., why do naive approaches fail?)
  - Why hasn't it been solved before? (Or, what's wrong with previous proposed solutions? How does mine differ?)
  - What are the key components of my approach and results? Also include any specific limitations.

Schedule & Process

- **Wednesday Schedule**
  - 3 sessions, ~20 minutes each
    - 6 people per group, seating suggestions provided
    - Check Google Forms [link] for classmates' paper choices
  - Introduce your research backgrounds and interests
  - Present your paper choices and analysis and comment on others
- Nominate interesting papers after class
  - Fill in the details of recommended papers through Google Sheets.
- **Find teammates for course project**
  - **3 people per group** by default
  - Fill in the names of team members in Google Sheets [link]

Schedule & Process

- **Friday Schedule**
  - TA will introduce some of the recommended papers on Friday
  - 2 project teams per group (6 people), seating suggestions provided
  - Develop your group project ideas through the lens of the 5 questions, and offer suggestions to others
  - Instructors will be around to provide advice.
  - Ask LLMs during your brainstorming process.
- **Grading for Paper Discussion & Teammate Finding**
  - Attendance (5%) – 3~4 lectures, missing each group discussion deduct 1%
  - Proposal writing (10%) – the 5 questions for your group project, more details will be shared next week