Graph Neural Networks: Theory

Jiaxuan You

Assistant Professor at UIUC CDS

CS598: Deep Learning with Graphs, 2024 Fall
**https://ulab-uiuc.github.io/CS598/**

Logistics: Coding Homework

- Colab Assignment 2 Out
  - Assignment will be released on Canvas today
  - Construct graph neural networks using PyTorch Geometric and apply the model on Open Graph Benchmark (OGB) datasets
  - Submit your code and written answers downloaded from Colab to Canvas by **Oct 13 (Sun) 11:59 PM, CT**.

Recap: A General GNN Framework
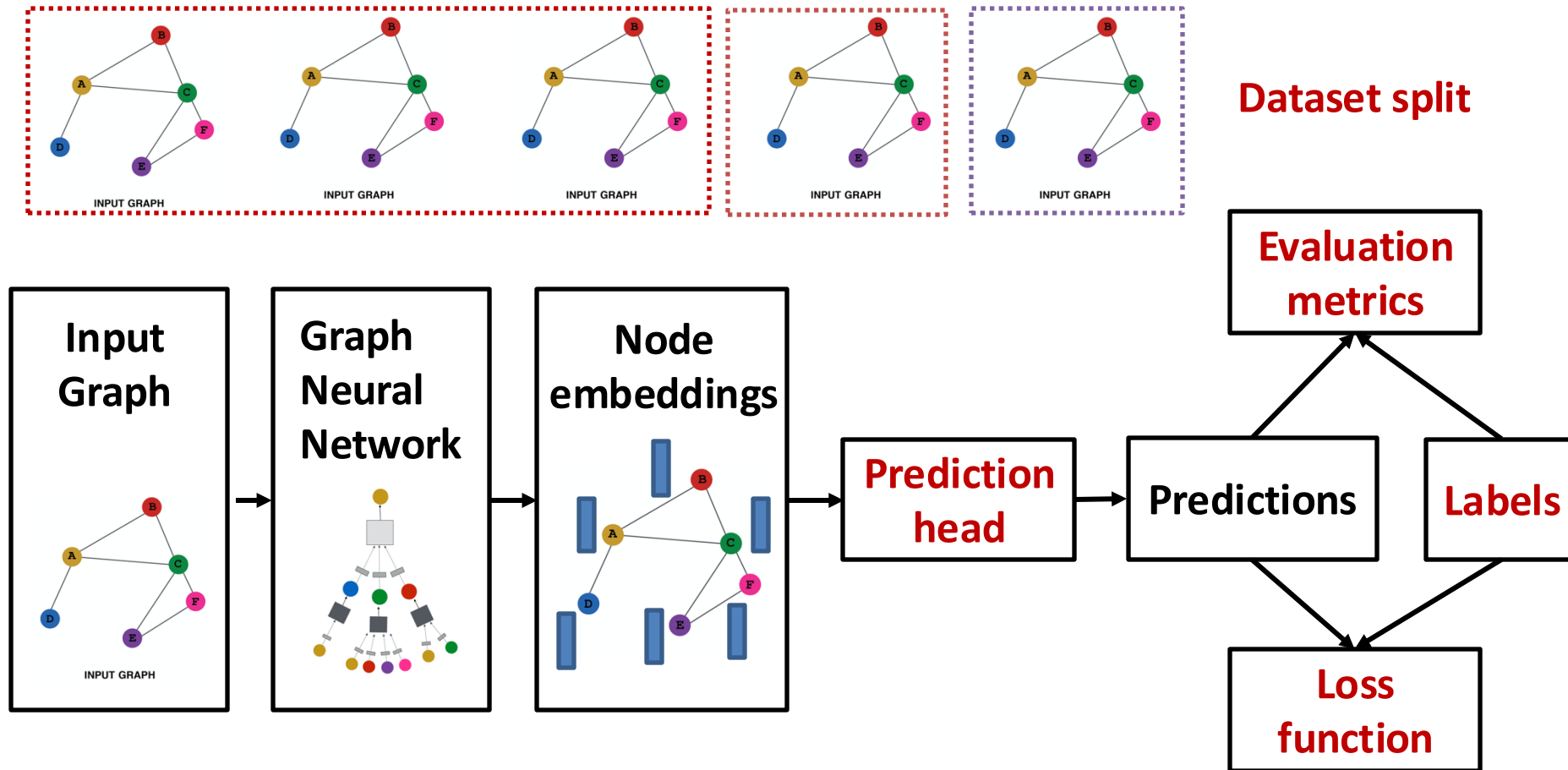
(5) Learning objective

GNN Layer 1

(2) Aggregation

(1) Message
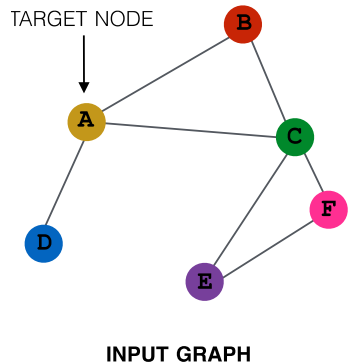
(3) Layer connectivity

GNN Layer 2
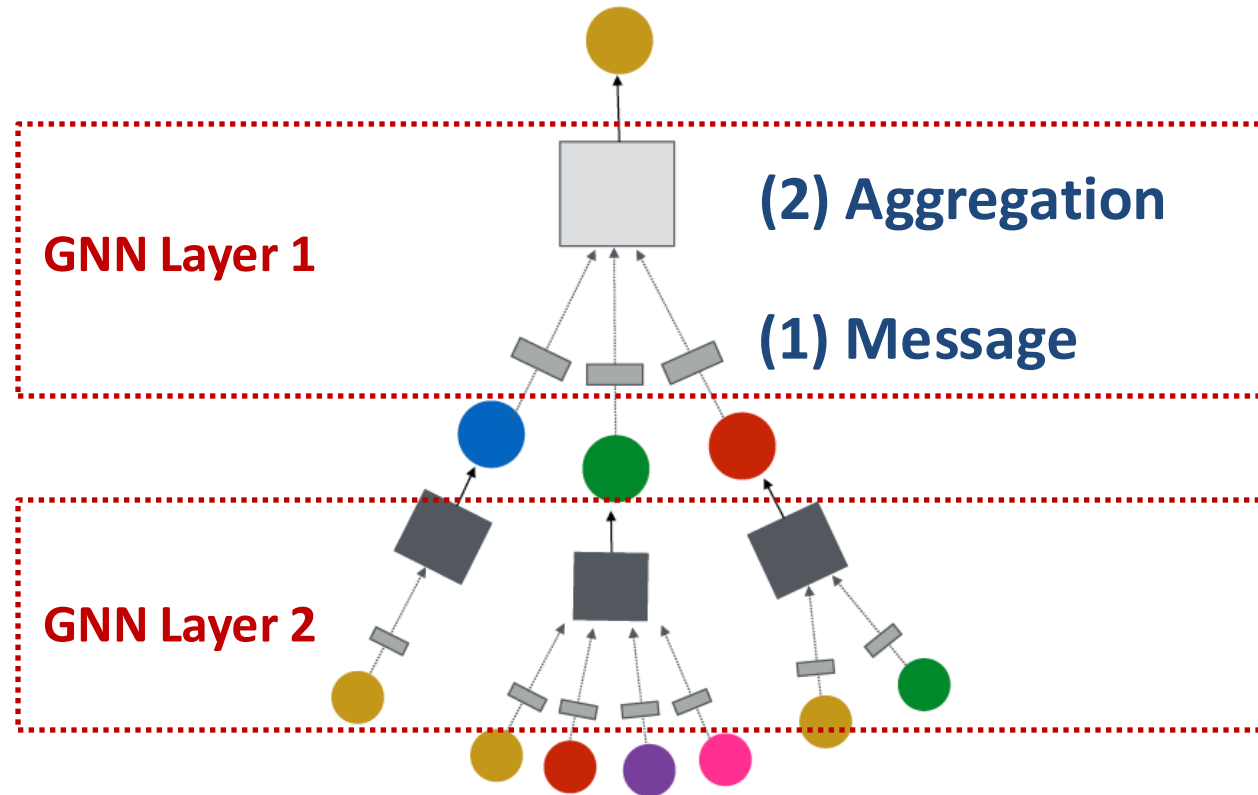
(4) Graph augmentation

TARGET NODE

INPUT GRAPH

Recap: GNN Training Pipeline

Implementation resources:

PyG provides core modules for this pipeline

GraphGym further implements the full pipeline to facilitate GNN design

Graph Neural Networks: Theory

How Expressive are Graph Neural Networks?

Theory of GNNs

How powerful are GNNs?

- **Many GNN models have been proposed (e.g., GCN, GAT, GraphSAGE, design space).**

- **What is the expressive power (ability to distinguish different graph structures) of these GNN models?**

- **How to design a maximally expressive GNN model?**
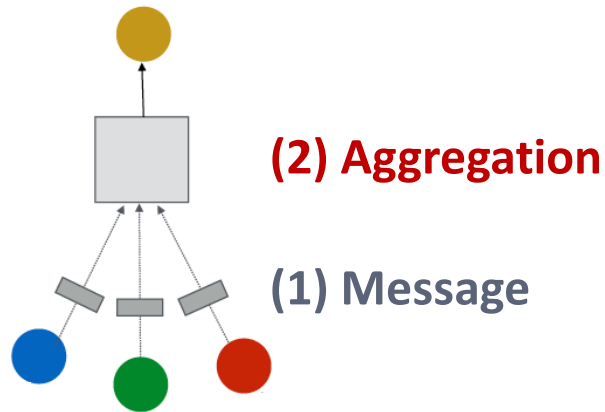
Background: A Single GNN Layer

- **We focus on message passing GNNs:**
  - **(1) Message**: each node computes a message
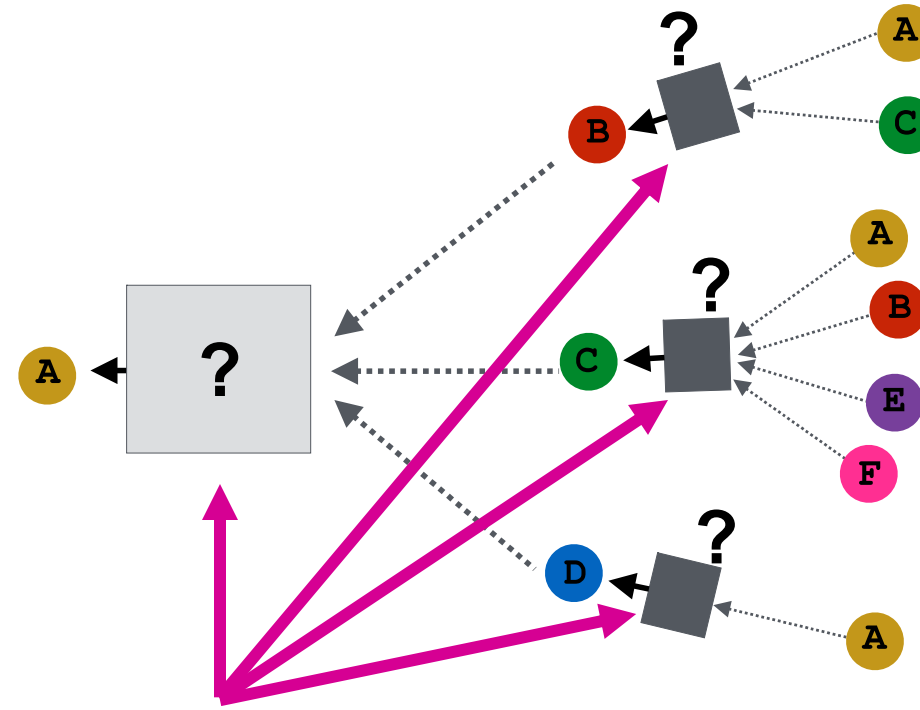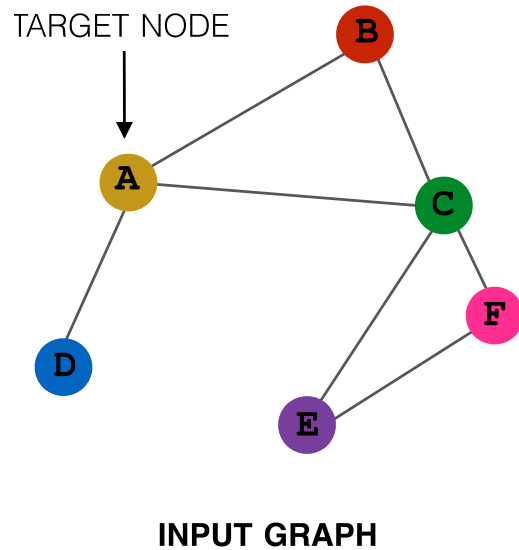 $$\mathbf{m}_u^{(l)} = \text{MSG}^{(l)}\left(\mathbf{h}_u^{(l-1)}\right), u \in \{N(v) \cup v\}$$
  - **(2) Aggregation**: aggregate messages from neighbors
  $$\mathbf{h}_v^{(l)} = \text{AGG}^{(l)}\left(\left\{\mathbf{m}_u^{(l)}, u \in N(v)\right\}, \mathbf{m}_v^{(l)}\right)$$

(2) Aggregation

(1) Message
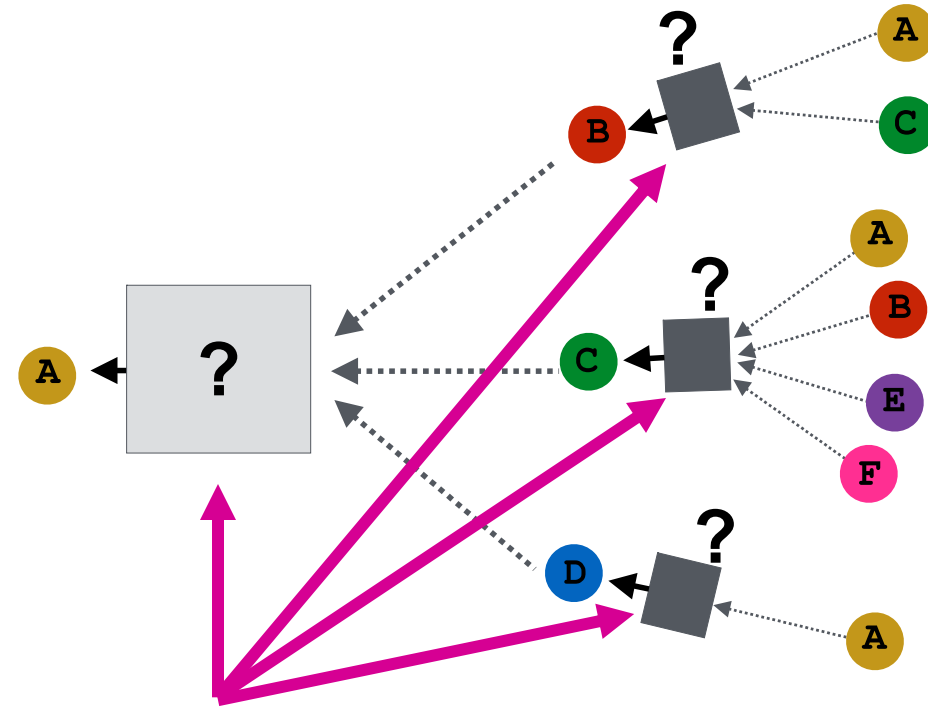
Background: Many GNN Models

- Many GNN models have been proposed:
    - GCN, GraphSAGE, GAT, Design Space etc.

TARGET NODE

INPUT GRAPH

Different GNN models use different neural networks in the box

GNN Model Example (1)

- **GCN (mean-pool)**

TARGET NODE

INPUT GRAPH
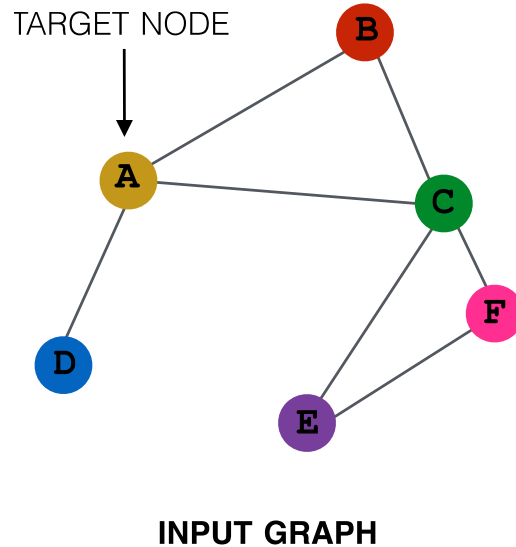
Element-wise mean pooling + Linear + ReLU non-linearity

GNN Model Example (2)

- **GraphSAGE (max-pool)**

TARGET NODE

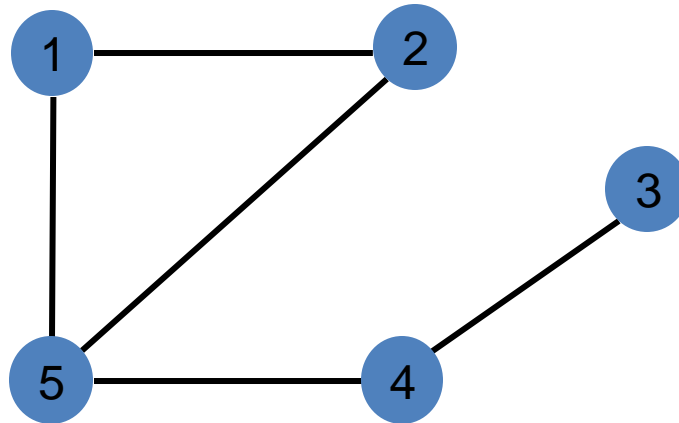INPUT GRAPH

?

?

?

?
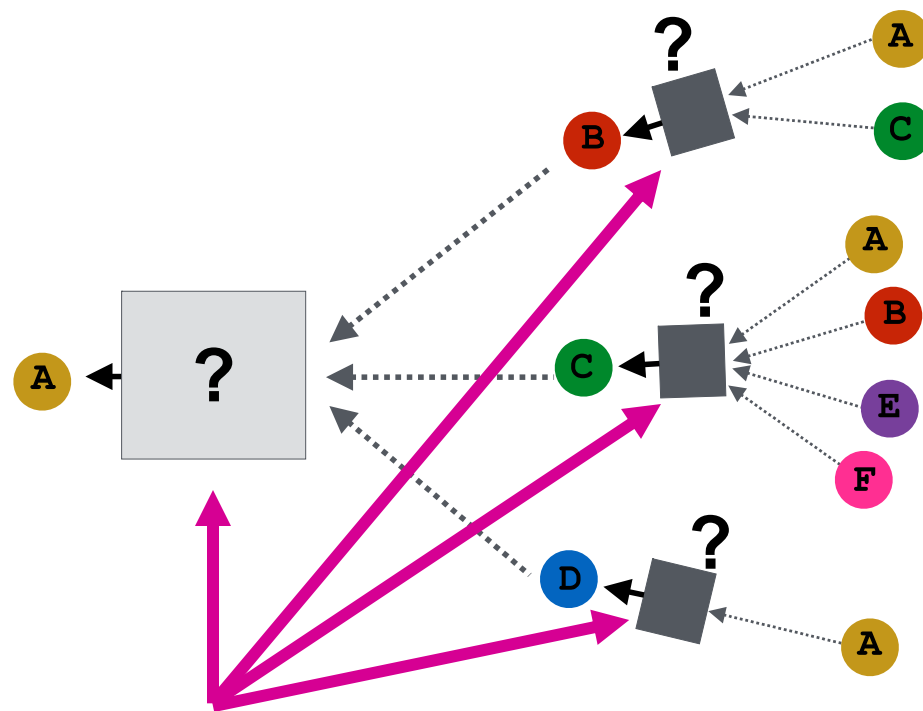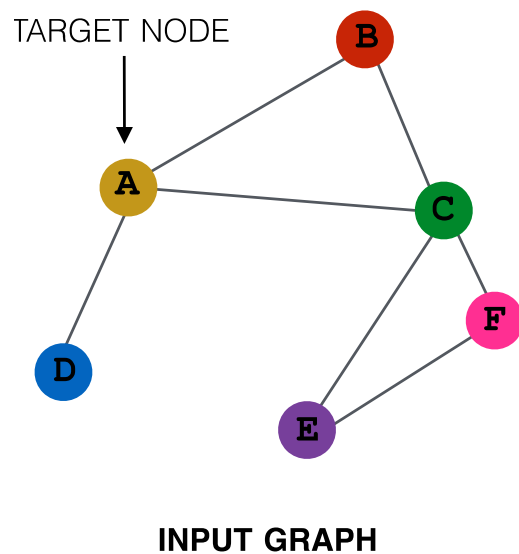
MLP + element-wise max-pooling

Note: Node Colors

- We use node same/different **colors** to represent nodes with same/different features.
  - For example, the graph below assumes all the nodes share the same feature.

- **Key question**: How well can a GNN distinguish different graph structures?

Local Neighborhood Structures

- We specifically consider **local neighborhood structures** around each node in a graph.

  - **Example:** Nodes 1 and 5 have **different** neighborhood structures because they have different node degrees.
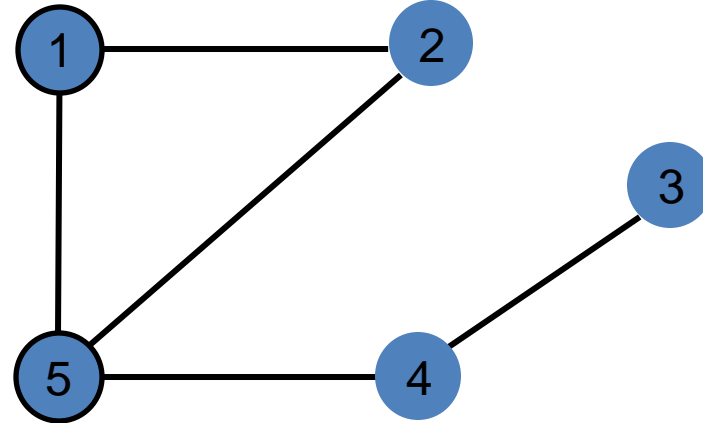
Local Neighborhood Structures

- We specifically consider **local neighborhood structures** around each node in a graph.

  - **Example:** Nodes 1 and 4 both have the same node degree of 2. However, they still have **different** neighborhood structures because **their neighbors have different node degrees.**

Node 1 has neighbors of degrees 2 and 3.
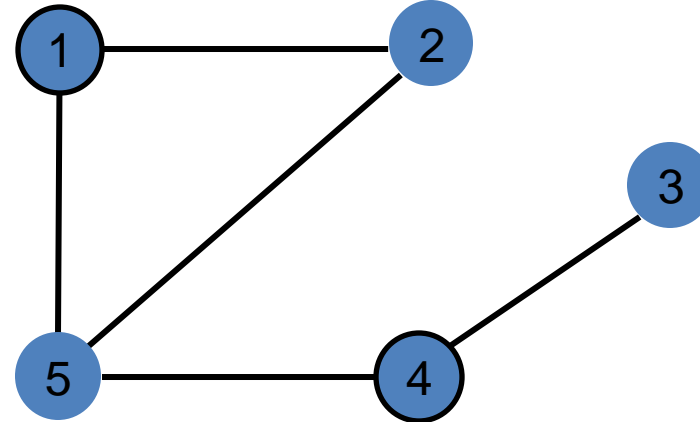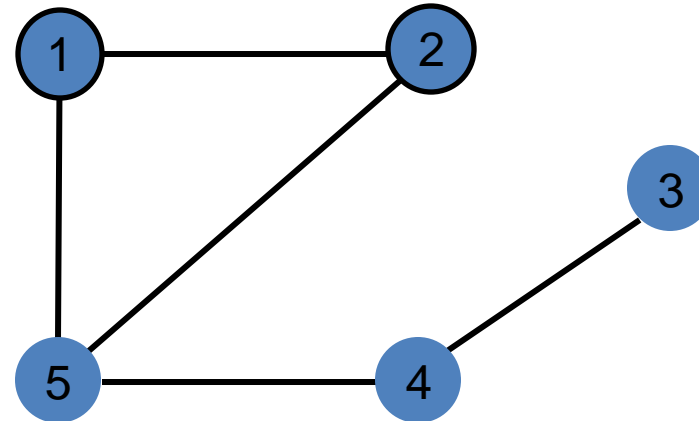Node 4 has neighbors of degrees 1 and 3.

Local Neighborhood Structures

- We specifically consider **local neighborhood structures** around each node in a graph.
  - **Example:** Nodes 1 and 2 have the **same** neighborhood structure because **they are symmetric within the graph.**

Node 1 has neighbors of degrees 2 and 3.
Node 2 has neighbors of degrees 2 and 3.
And even if we go a step deeper to 2nd hop neighbors, both nodes have the same degrees (Node 4 of degree 2)

Local Neighborhood Structures

- **Key question**: **Can GNN node embeddings distinguish different node's local neighborhood structures?**

  - If so, when? If not, when will a GNN fail?

- **Next**: **We need to understand how a GNN captures local neighborhood structures.**

  - **Key concept**: **Computational graph**

Computational Graph (1)

- **In each layer, a GNN aggregates neighboring node embeddings.**

- A GNN generates node embeddings through a computational graph defined by the neighborhood.

  - **Ex:** Node 1's computational graph (2-layer GNN)
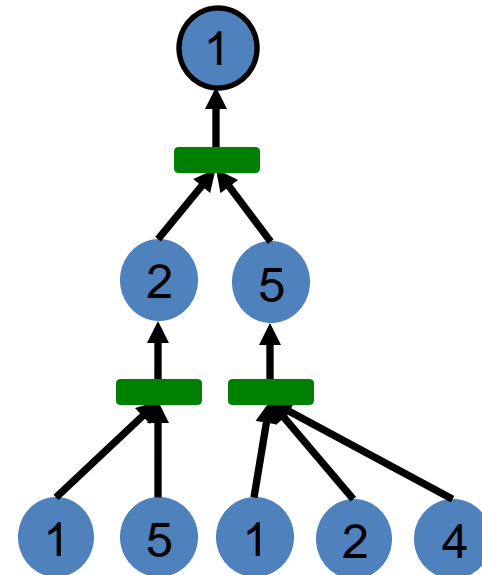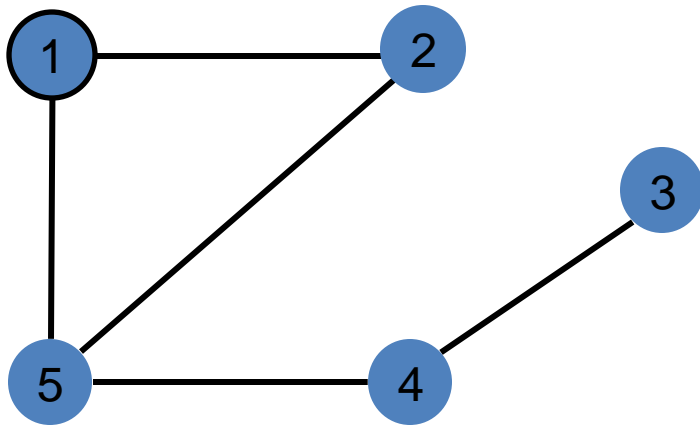
Computational Graph (2)

- **Ex:** Nodes 1 and 2's computational graphs.
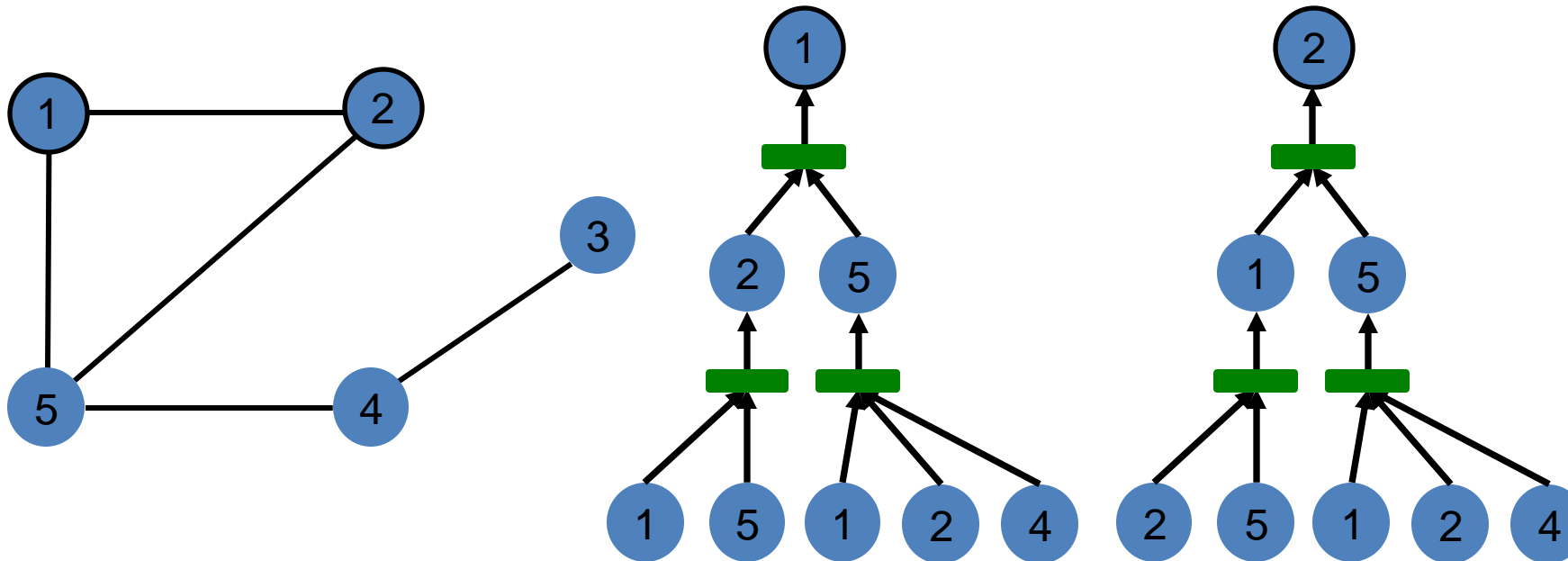
Computational Graph (3)

- **Ex:** Nodes 1 and 2's computational graphs.
  - **But GNN only sees node features (not IDs):**
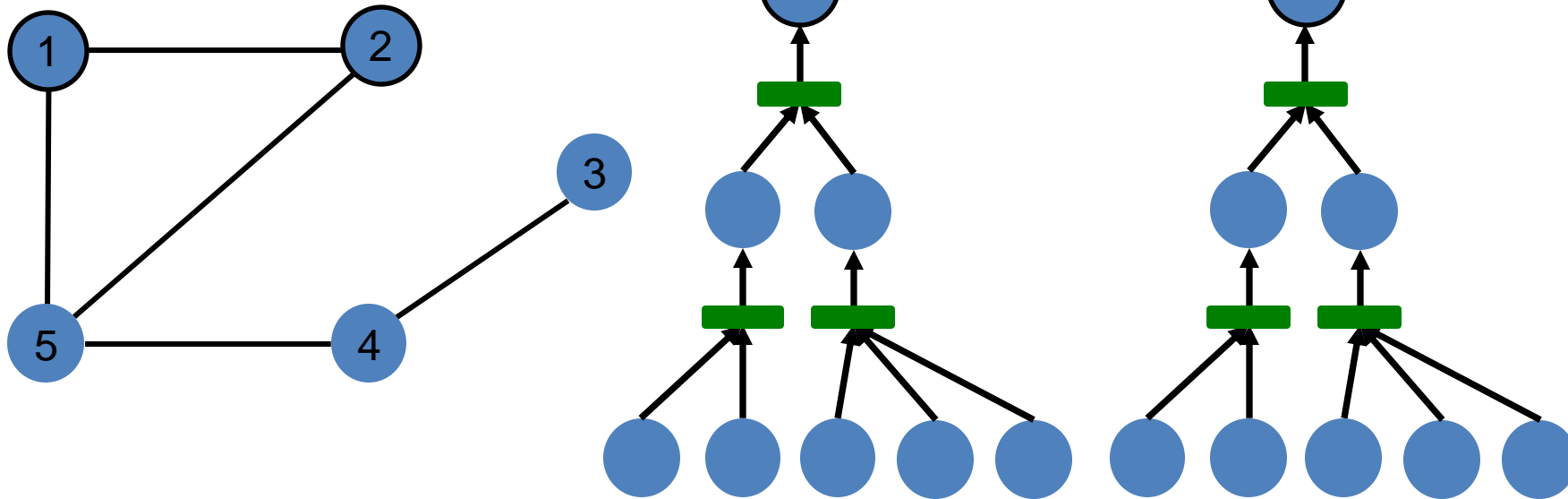
Computational Graph (4)

- **A GNN will generate <u>the same embedding</u> for nodes 1 and 2 because:**
  - Computational graphs are the same.
  - Node features (colors) are identical.
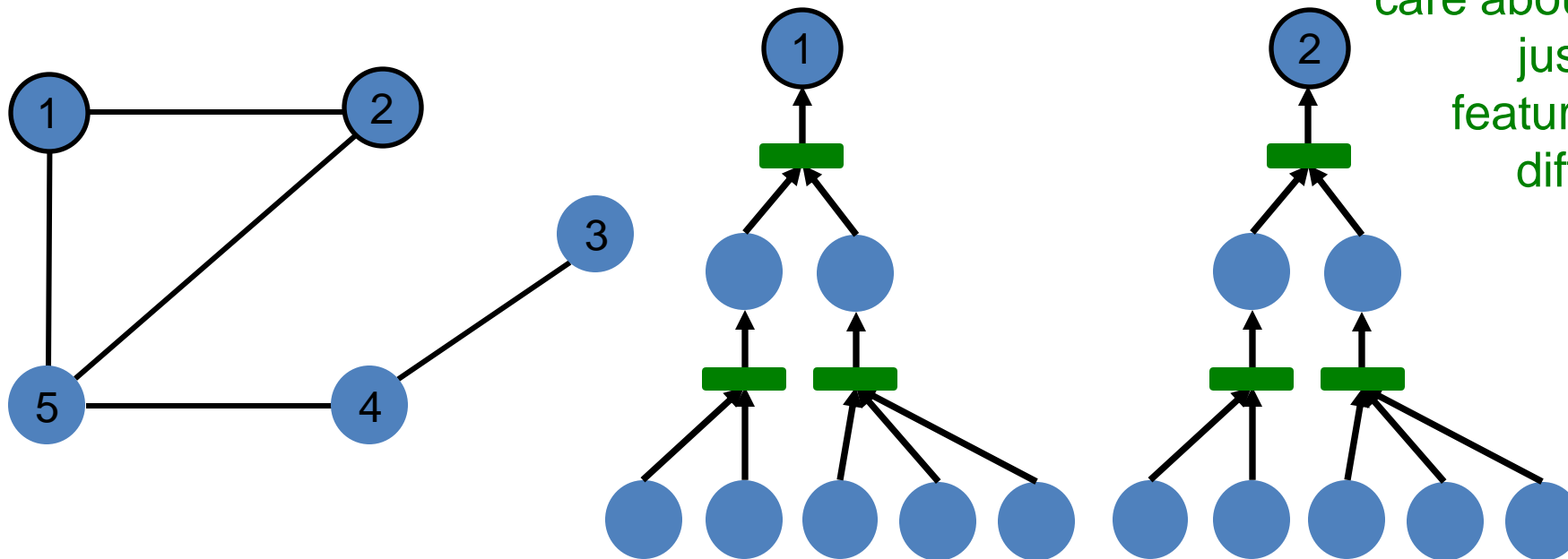
Note: GNN does not care about node ids, it just aggregates features vectors of different nodes.

GNN won't be able to distinguish nodes 1 and 2

Computational Graph

- **In general, different local neighborhoods define different computational graphs**

CS598: Deep Learning with Graphs, Jiaxuan You
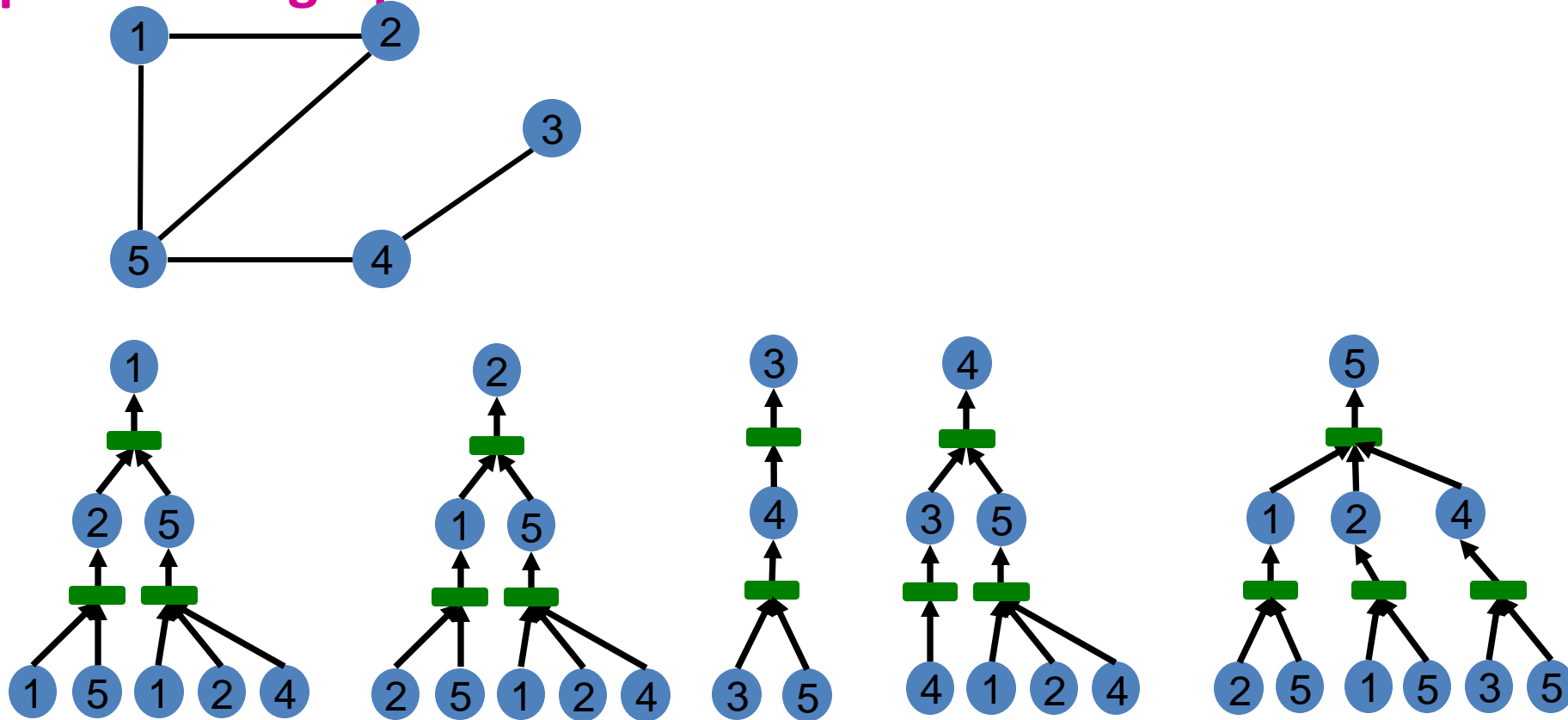
Computational Graph

- Computational graphs are identical to **rooted subtree structures** around each node.

Rooted subtree structures
(defined by recursively unfolding neighboring nodes from the root nodes)
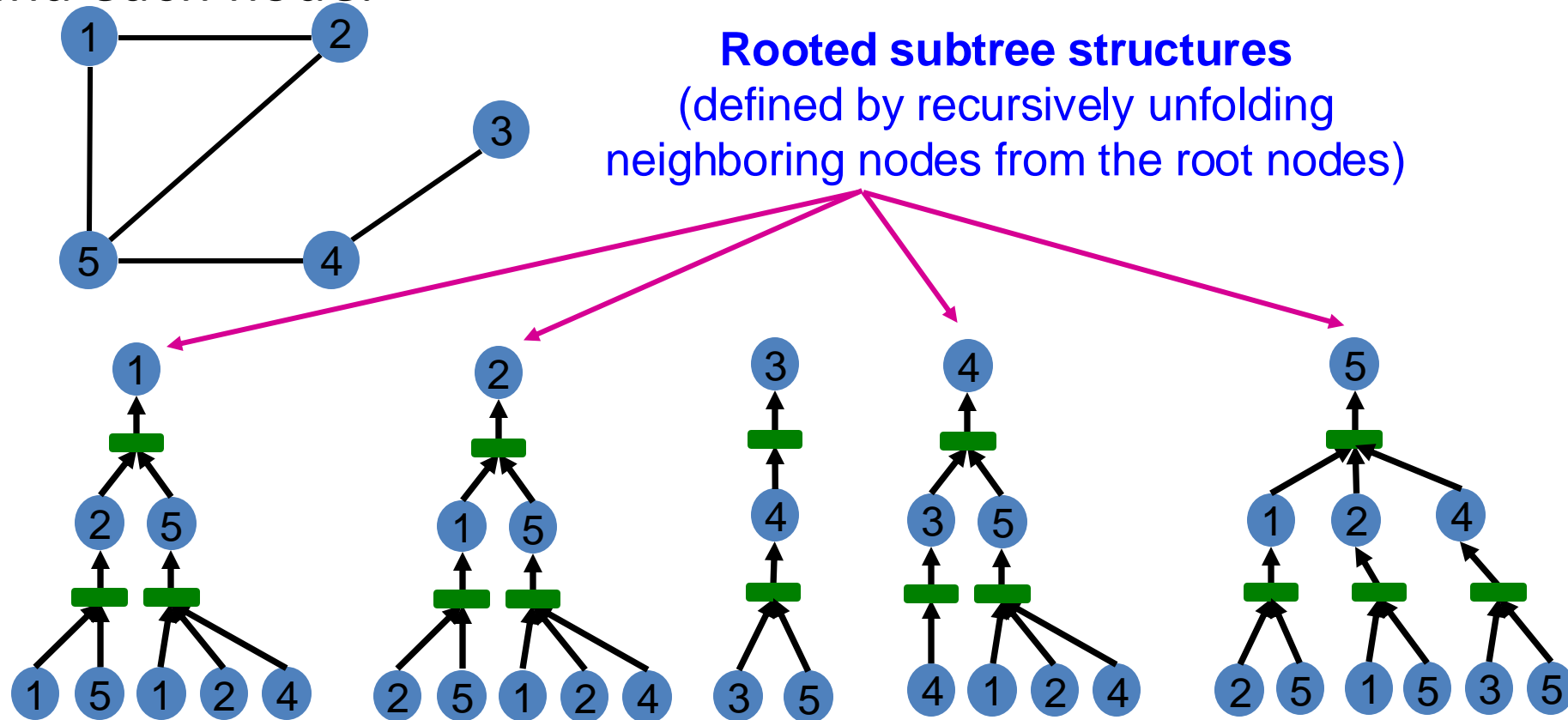
Computational Graph

- GNN's node embeddings capture rooted subtree structures.
- Most expressive GNN maps different rooted subtrees into different node embeddings (represented by different colors).
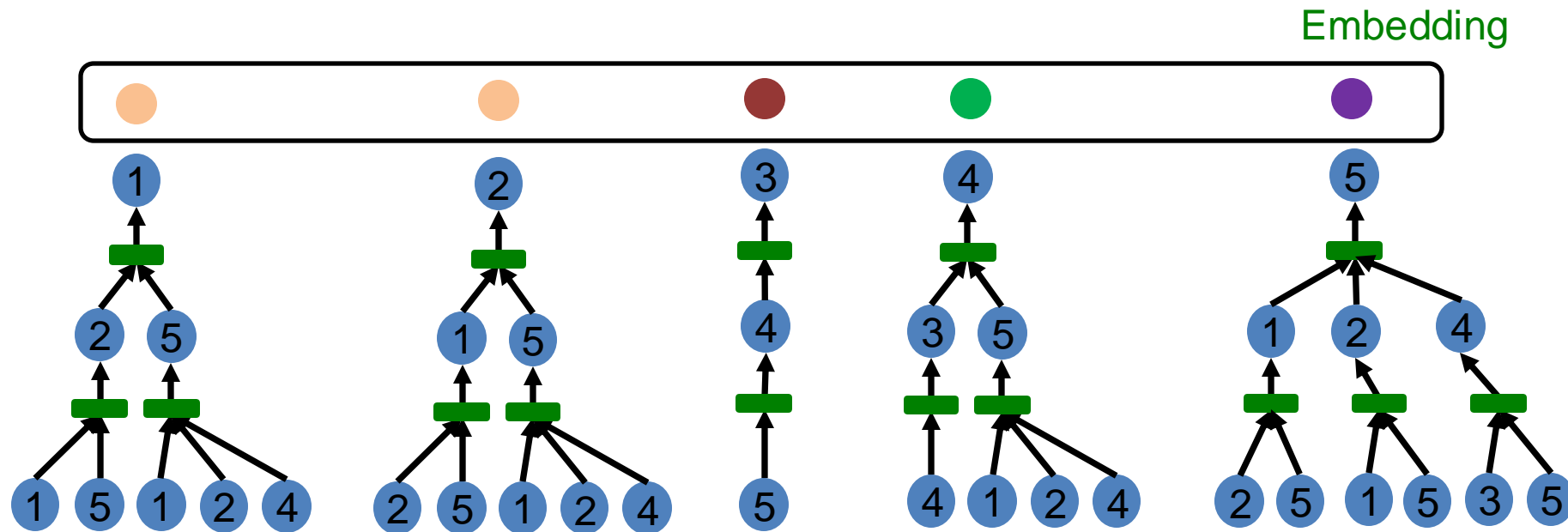
Recall: Injective Function

- **Function** $f: X \rightarrow Y$ is **injective** if it maps different elements into different outputs.

- **Intuition:** f retains all the information about input.
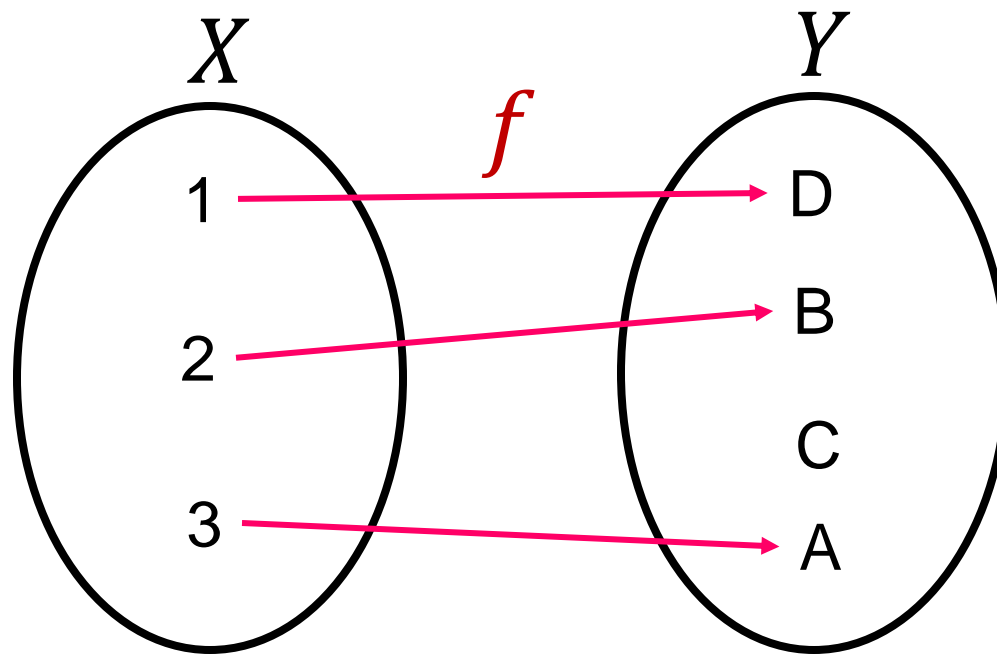
How Expressive is a GNN?

■ Most expressive *(message-passing)* GNN should map subtrees to the node embeddings **injectively**.

CS598: Deep Learning with Graphs, Jiaxuan You
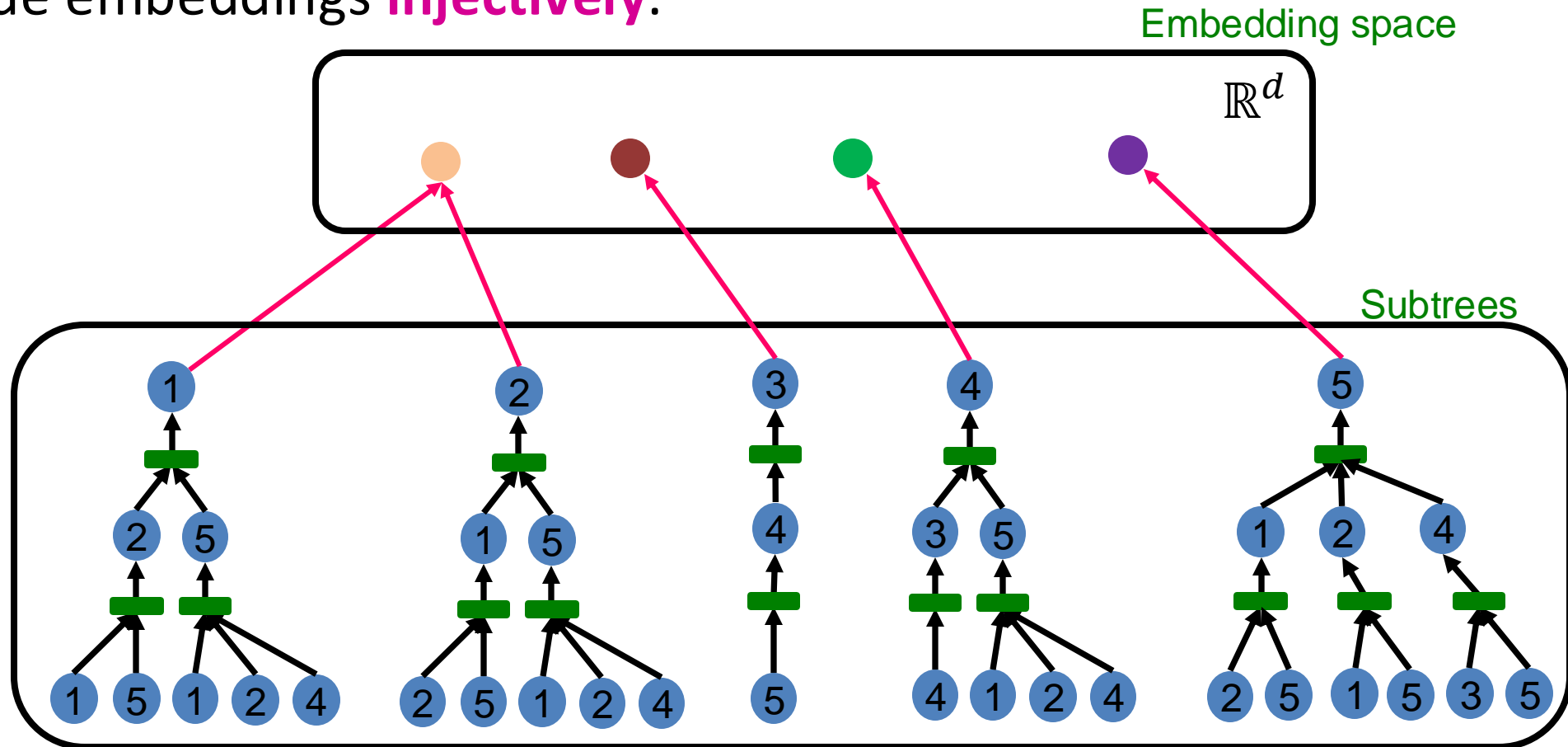
How Expressive is a GNN?

- **Key observation**: Subtrees of the same depth can be recursively characterized from the leaf nodes to the root nodes.
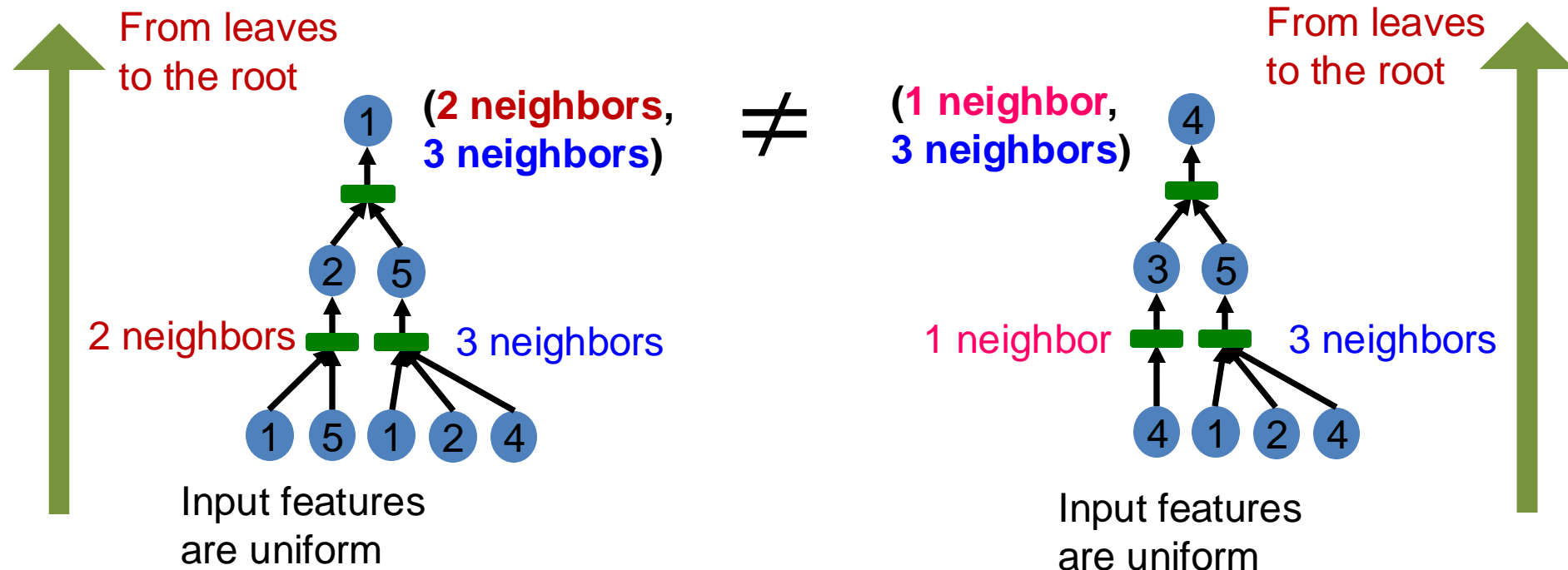
How Expressive is a GNN?

- If each step of GNN's aggregation **can fully retain the neighboring information**, the generated node embeddings can distinguish different rooted subtrees.

CS598: Deep Learning with Graphs, Jiaxuan You
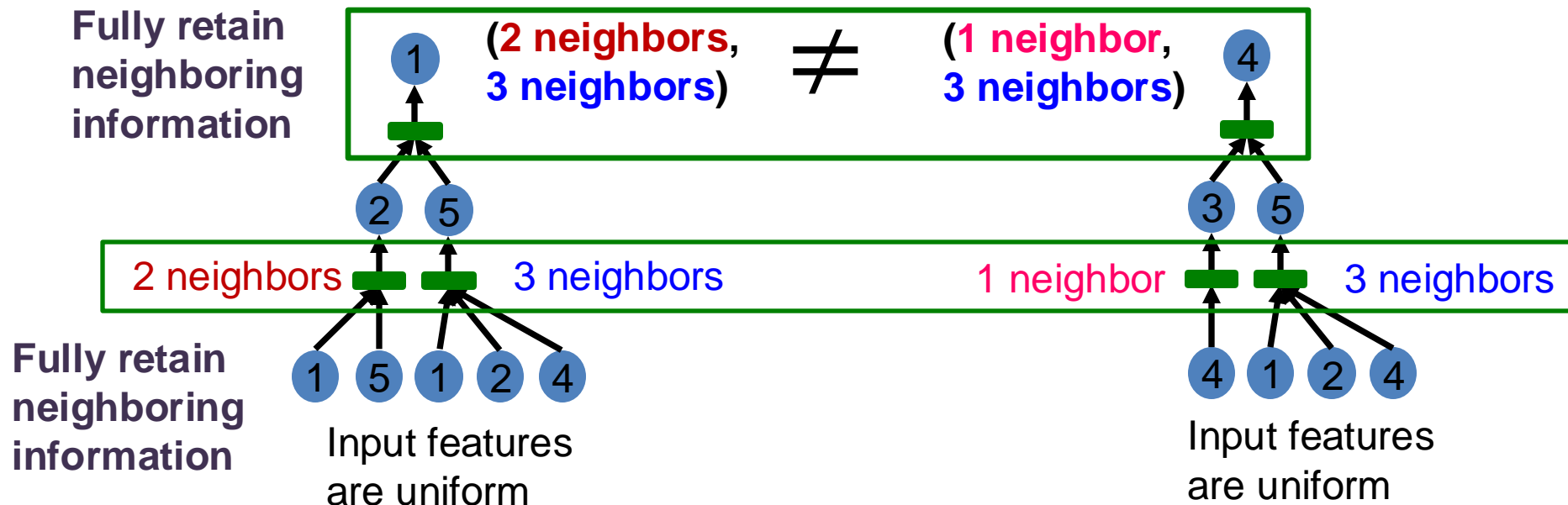
How Expressive is a GNN?

- In other words, most expressive GNN would use an **injective neighbor aggregation** function at each step.
  - Maps different neighbors to different embeddings.

Input features are uniform

Input features are uniform

CS598: Deep Learning with Graphs, Jiaxuan You
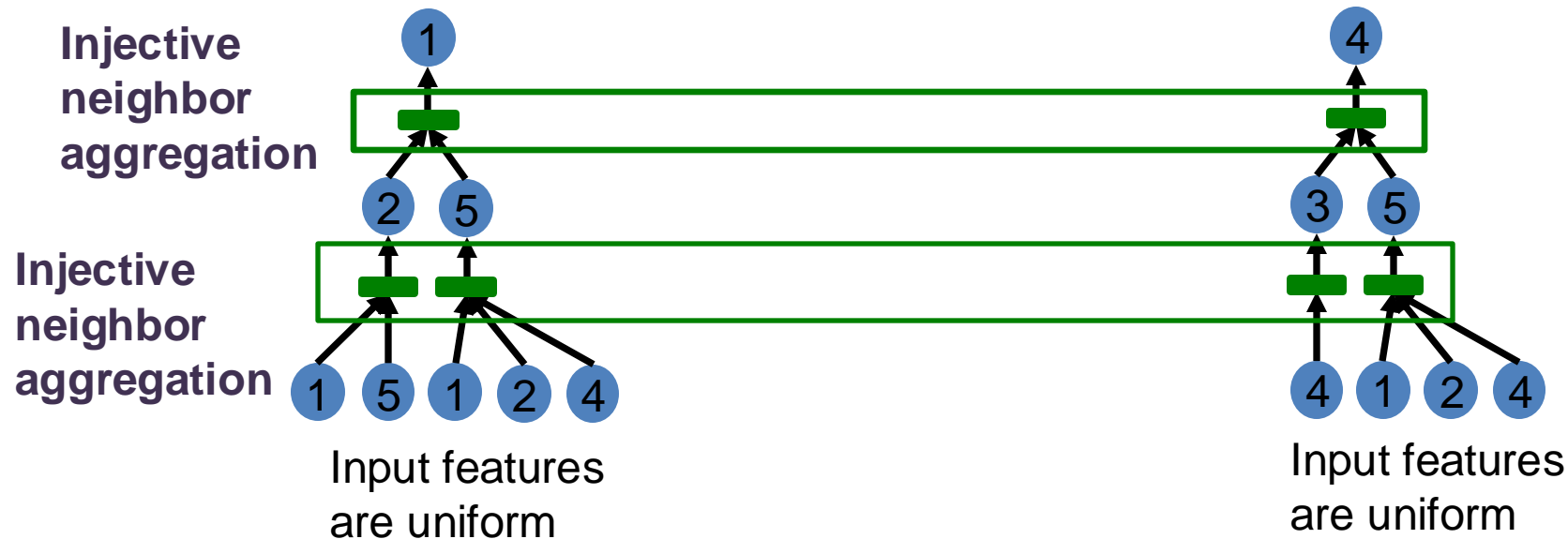
How Expressive is a GNN?

- **Summary so far**
  - To generate a node embedding, GNNs use a computational graph corresponding to a subtree rooted around each node.

Input graph
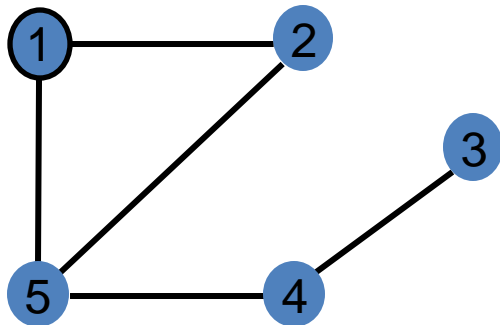
Computational graph = Rooted subtree
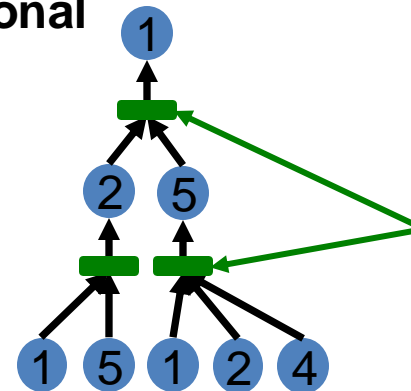
Using injective neighbor aggregation → distinguish different subtrees

  - GNN can fully distinguish different subtree structures if **every step of its neighbor aggregation is injective.**

Graph Neural Networks: Theory

Designing the Most Powerful Graph Neural Network

Expressive Power of GNNs

- **Key observation**: **Expressive power of GNNs can be characterized by that of neighbor aggregation functions they use.**

  - A more expressive aggregation function leads to a more expressive a GNN.

  - Injective aggregation function leads to the most expressive GNN.

- **Next:**

  - Theoretically analyze expressive power of aggregation functions.

Neighbor Aggregation

- **Observation**: **Neighbor aggregation** can be abstracted as **a function over a multi-set** (a set with repeating elements).

Neighbor aggregation

Equivalent

Multi-set function

Examples of multi-set

Same color indicates the same features.
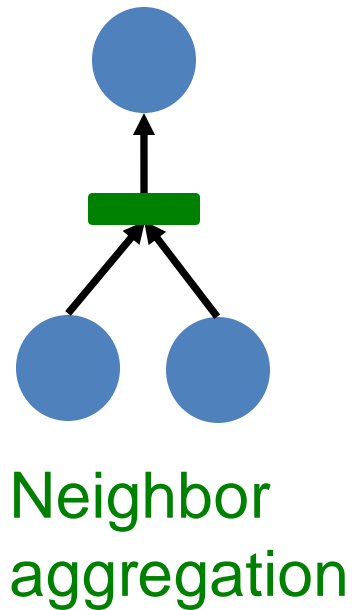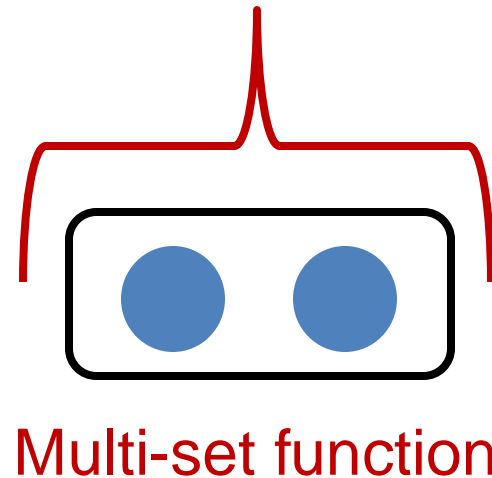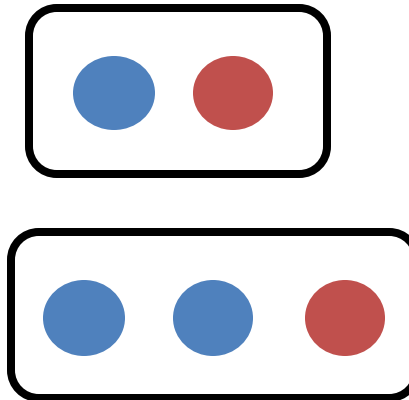
Neighbor Aggregation

- **Next**: **We analyze aggregation functions of two popular GNN models**
  - **GCN** (mean-pool) [Kipf & Welling, ICLR 2017]
    - Uses **element-wise** mean pooling over neighboring node features
    $$\text{Mean}(\{x_u\}_{u \in N(v)})$$
  - **GraphSAGE** (max-pool) [Hamilton et al. NeurIPS 2017]
    - Uses **element-wise** max pooling over neighboring node features
    $$\text{Max}(\{x_u\}_{u \in N(v)})$$

Neighbor Aggregation: Case Study

- **GCN (mean-pool)** [Kipf & Welling ICLR 2017]
  - Take **element-wise mean**, followed by linear function and ReLU activation, i.e., $\max(0, x)$.
  - **Theorem** [Xu et al. ICLR 2019]
    - GCN's aggregation **function cannot distinguish different multi-sets with the same color proportion**.

Failure case

- **Why?**

CS598: Deep Learning with Graphs, Jiaxuan You
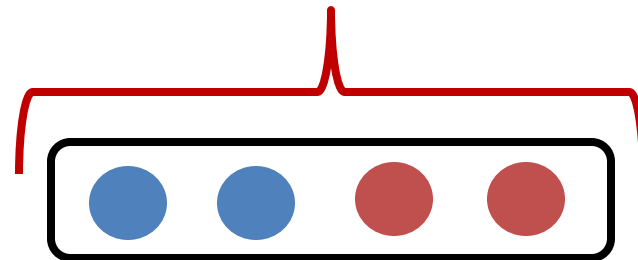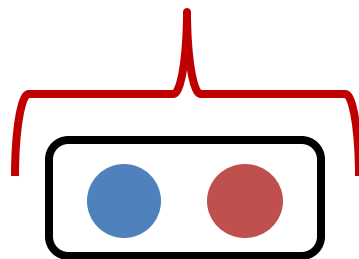
Neighbor Aggregation

- For simplicity, we assume node features (colors) are represented by **one-hot encoding**.

  - **Example:** If there are two distinct colors:

$$\bullet = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \qquad \bullet = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

  - This assumption is sufficient to illustrate how GCN fails.

Neighbor Aggregation: Case Study

- **GCN (mean-pool)** [Kipf & Welling ICLR 2017]

  - **Failure case illustration**

Same outputs!

Linear + ReLU   Linear + ReLU

Element-wise-mean-pool

$$\begin{pmatrix} 0.5 \\ 0.5 \end{pmatrix}$$

$$\begin{pmatrix} 0.5 \\ 0.5 \end{pmatrix}$$

$$\begin{pmatrix} 1 \\ 0 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

$$\begin{pmatrix} 1 \\ 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$
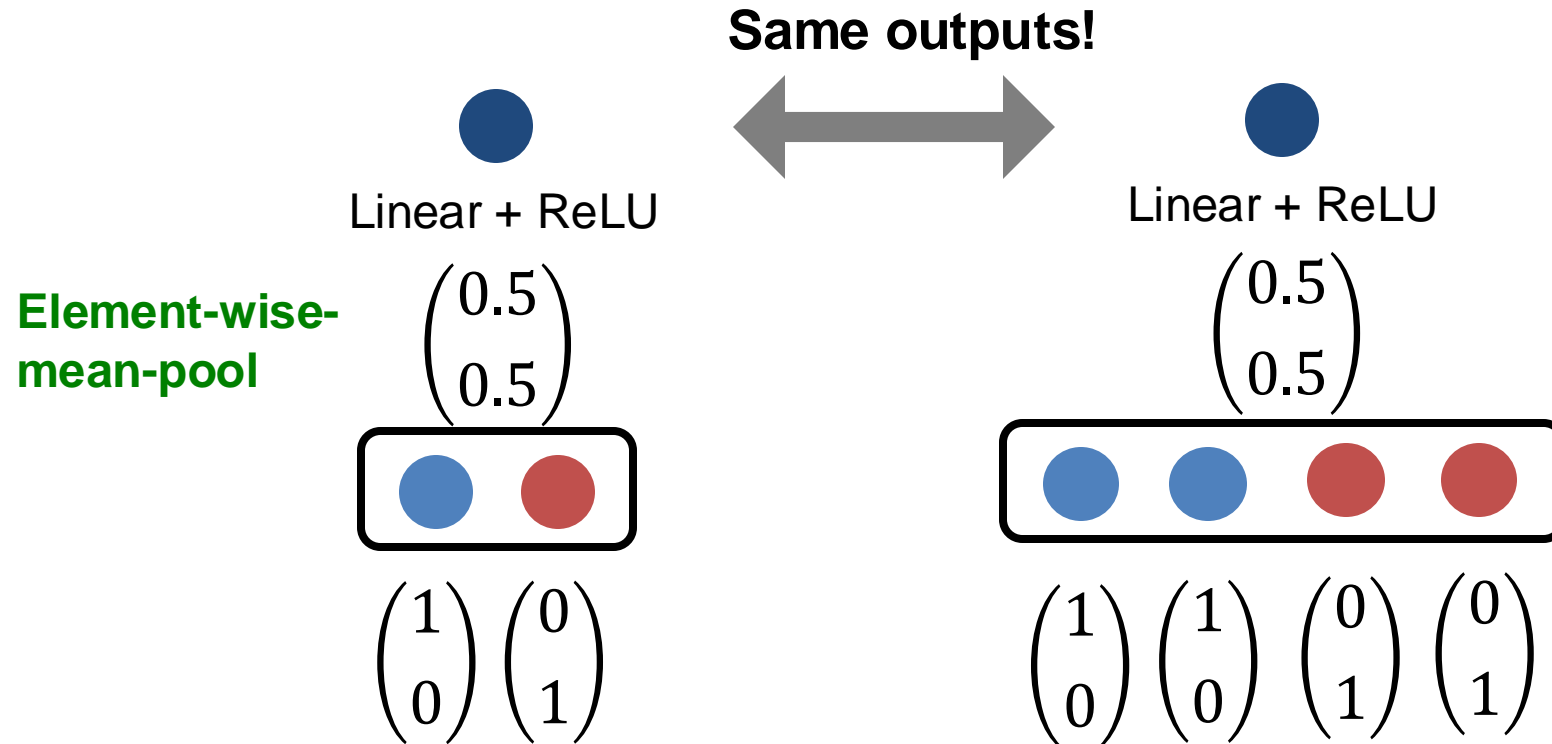
Neighbor Aggregation: Case Study

- **GraphSAGE (max-pool)** [Hamilton et al. NeurIPS 2017]

  - Apply an MLP, then take **element-wise max**.

  - **Theorem** [Xu et al. ICLR 2019]

    - GraphSAGE's aggregation **function cannot distinguish different multi-sets with the same set of distinct colors**.

      Failure case
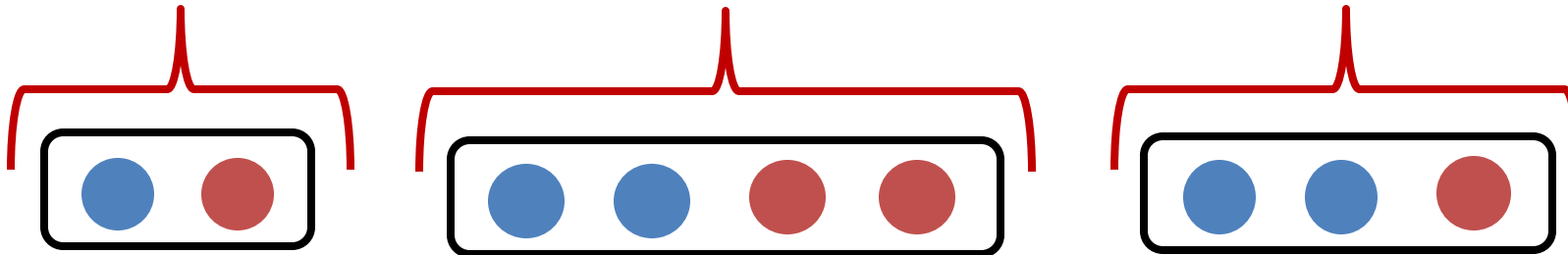
- **Why?**

Neighbor Aggregation: Case Study

- **GraphSAGE (max-pool)** [Hamilton et al. NeurIPS 2017]

  - **Failure case illustration**

The same outputs!

Element-wise-max-pool

$$\begin{pmatrix} 1 \\ 1 \end{pmatrix} \longleftrightarrow \begin{pmatrix} 1 \\ 1 \end{pmatrix} \longleftrightarrow \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

For simplicity, assume the one-hot encoding after **MLP**.

$$\begin{pmatrix} 1 \\ 0 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} \qquad \begin{pmatrix} 1 \\ 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} \qquad \begin{pmatrix} 1 \\ 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

MLP
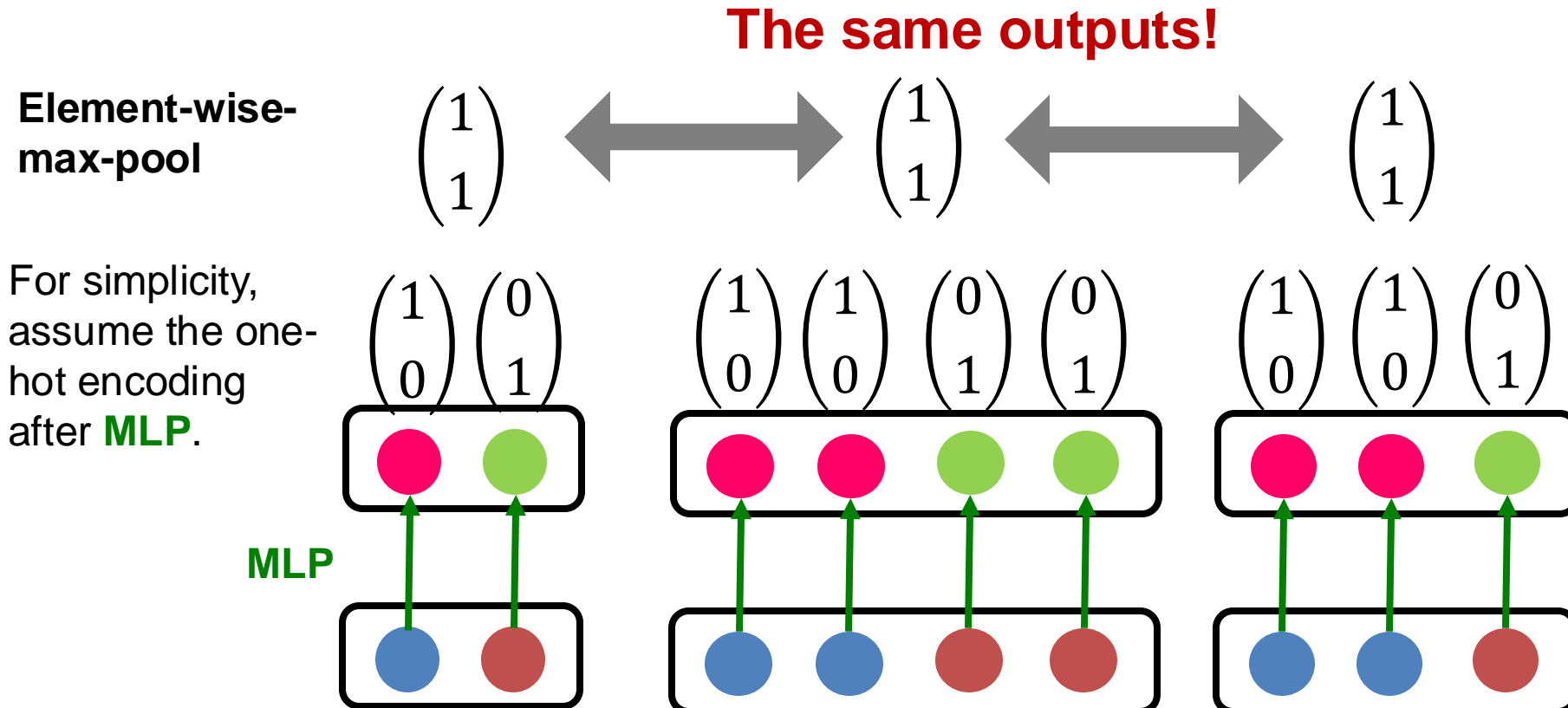
Summary So Far

- We analyzed the **expressive power of GNNs.**

- **Main takeaways**:

  - Expressive power of GNNs can be characterized by that of the neighbor aggregation function.

  - Neighbor aggregation is a function over multi-sets (sets with repeating elements)

  - GCN and GraphSAGE's aggregation functions fail to distinguish some basic multi-sets; hence **not injective**.

  - Therefore, GCN and GraphSAGE are **not** maximally powerful GNNs.

Designing Most Expressive GNNs

- **Our goal**: **Design maximally powerful GNNs in the class of message-passing GNNs.**

- This can be achieved by designing **injective** **neighbor aggregation function over multi-sets**.

- **Here, we design a neural network that can model injective multiset function.**

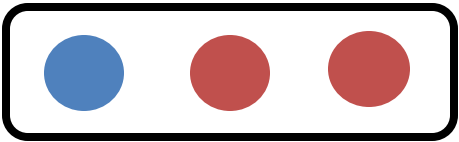# Injective Multi-Set Function

Theorem [Xu et al. ICLR 2019]

Any injective multi-set function can be expressed as:

Some non-linear function

Some non-linear function

$$\Phi\left(\sum_{x \in S} f(x)\right)$$

Sum over multi-set

S : multi-set

$$\Phi\left(f(\bullet) + f(\bullet) + f(\bullet)\right)$$

Injective Multi-Set Function

Proof Intuition: [Xu et al. ICLR 2019]

f produces one-hot encodings of colors. Summation of the one-hot encodings retains all the information about the input multi-set.

$$\Phi\left(\sum_{x \in S} f(x)\right)$$

Example:

$$\Phi\left[f\left[\bigcirc\right] + f\left[\bigcirc\right] + f\left[\bigcirc\right]\right]$$

One-hot $\begin{pmatrix} 1 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$

Universal Approximation Theorem

- **How to model Φ and f in $\Phi\left(\sum_{x \in S} f(x)\right)$?**

- We use a Multi-Layer Perceptron (MLP).

- **Theorem: Universal Approximation Theorem** [Hornik et al., 1989]

  - 1-hidden-layer MLP with sufficiently-large hidden dimensionality and appropriate non-linearity $\sigma(\cdot)$ (including ReLU and sigmoid) can **approximate any continuous function to an arbitrary accuracy**.
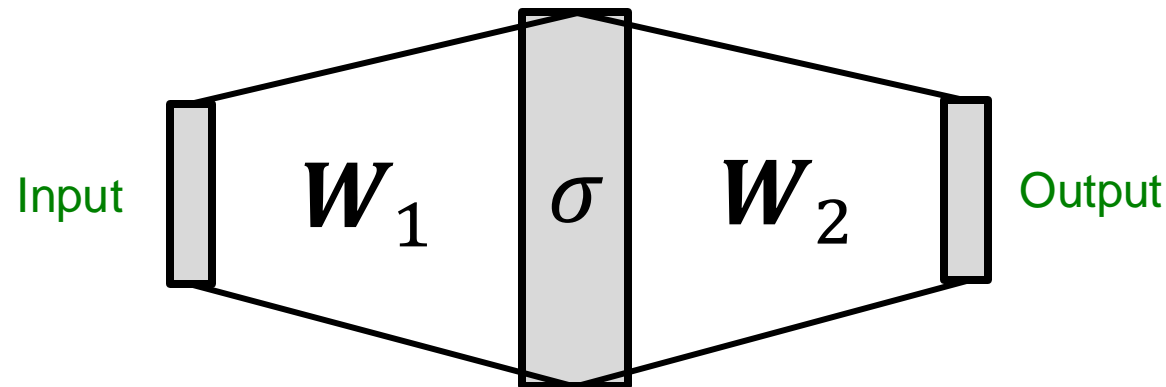
Injective Multi-Set Function

- **We have arrived at a neural network that can model any injective multiset function.**

$$\mathrm{MLP}_{\Phi}\left(\sum_{x \in S} \mathrm{MLP}_f(x)\right)$$

  - In practice, MLP hidden dimensionality of 100 to 500 is sufficient.

Most Expressive GNN

- **Graph Isomorphism Network (GIN)** [Xu et al. ICLR 2019]

  - Apply an MLP, element-wise **sum**, followed by another MLP.

$$\mathrm{MLP}_\Phi \left(\sum_{x \in S} \mathrm{MLP}_f(x) \right)$$

- **Theorem** [Xu et al. ICLR 2019]

  - GIN's neighbor aggregation function is injective.

- **No failure cases!**

- **GIN is THE most expressive GNN** in the class of message-passing GNNs we have introduced!

Full Model of GIN

- **So far: We have described the neighbor aggregation part of GIN.**

- We now describe the full model of GIN by relating it to **WL graph kernel** (traditional way of obtaining graph-level features).
  - We will see how GIN is a "neural network" version of the WL graph kernel.

Relation to WL Graph Kernel

Color refinement algorithm in WL kernel.

- **Given:** A graph G with a set of nodes V.

  - Assign an initial color $c^{(0)}(v)$ to each node v.

  - Iteratively refine node colors by

  $$c^{(k+1)}(v) = \text{HASH}\left(c^{(k)}(v), \left\{c^{(k)}(u)\right\}_{u \in N(v)}\right),$$
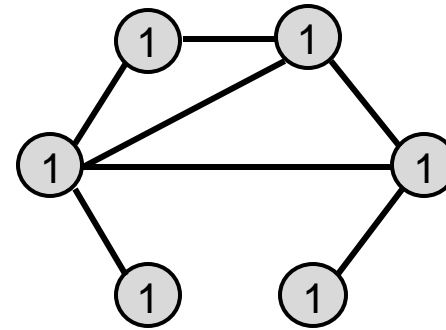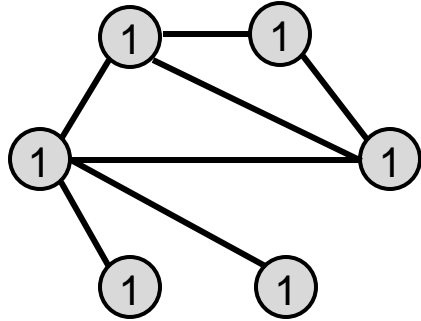
  where HASH maps different inputs to different colors.

  - After K steps of color refinement, $c^{(K)}(v)$ summarizes the structure of K-hop neighborhood
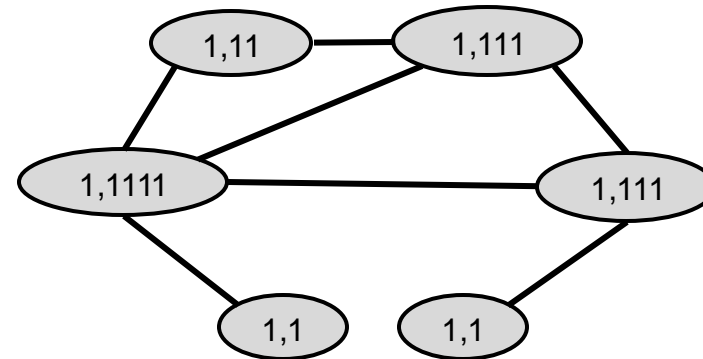
Color Refinement (1)
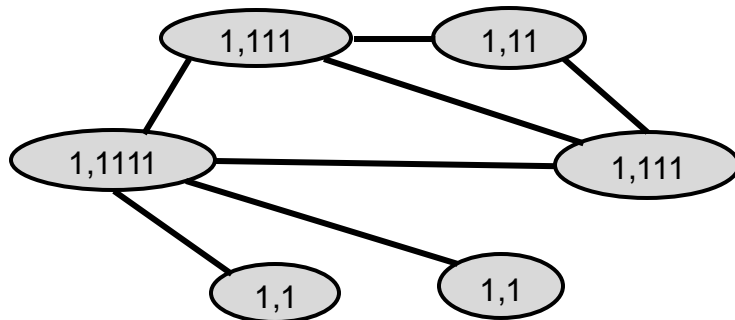
Example of color refinement given <u>two graphs</u>

- Assign initial colors

- Aggregate neighboring colors

Color Refinement (2)

Example of color refinement given two graphs

- Aggregated colors:
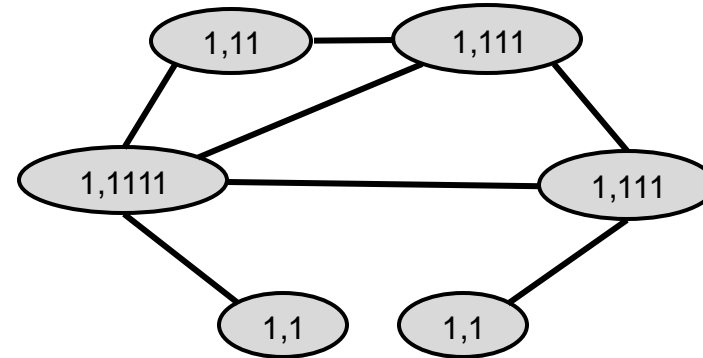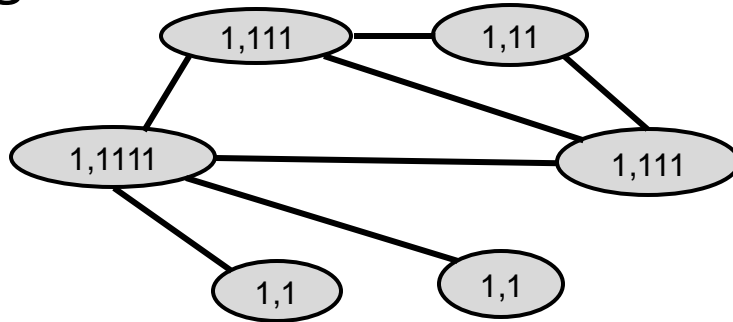
- **Injectively** HASH the aggregated colors
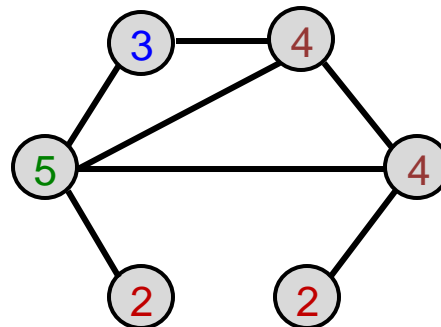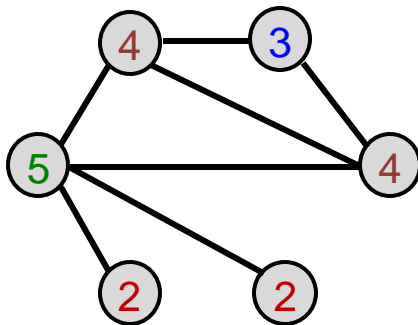
HASH table: **Injective!**

1,1	-->	2
1,11	-->	3
1,111	-->	4
1,1111	-->	5

Color Refinement (3)

Example of color refinement given two graphs

- Process continues until a stable coloring is reached / 2 graphs can be differentiated

- Two graphs are considered **isomorphic** if they have the same set of colors.
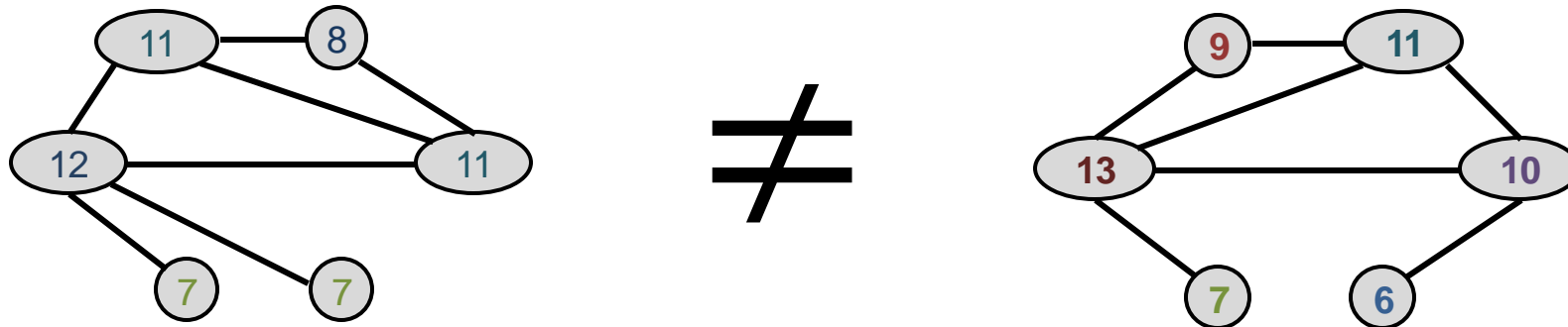
The Complete GIN Model

- **GIN uses a neural network to model the injective HASH function.**

$$c^{(k+1)}(v) = \text{HASH}\left(c^{(k)}(v), \left\{c^{(k)}(u)\right\}_{u \in N(v)}\right)$$

- Specifically, we will model the injective function over the tuple:

$$\left(c^{(k)}(v), \left\{c^{(k)}(u)\right\}_{u \in N(v)}\right)$$

Root node
features

Neighboring
node colors

The Complete GIN Model

Theorem (Xu et al. ICLR 2019)

Any injective function over the tuple

Root node feature $\left(\boxed{c^{(k)}(v)}, \boxed{\left\{c^{(k)}(u)\right\}_{u \in N(v)}}\right)$ Neighboring node features

can be modeled as

$$\text{MLP}_\Phi\left((1 + \epsilon) \cdot \text{MLP}_f(c^{(k)}(v))) + \sum_{u \in N(v)} \text{MLP}_f(c^{(k)}(u))\right)$$

where ϵ is a learnable scalar.

The Complete GIN Model

- If input feature $c^{(0)}(v)$ is represented as one-hot, **direct summation is injective**.

Example: $\Phi \left(\text{⬤} + \text{⬤} + \text{⬤} \right)$

$$\begin{pmatrix} 1 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$$

- We only need Φ to ensure the injectivity.

$$\text{GINConv}\left(c^{(k)}(v), \left\{ c^{(k)}(u) \right\}_{u \in N(v)} \right) = \text{MLP}_\Phi \left((1 + \epsilon) \cdot c^{(k)}(v) + \sum_{u \in N(v)} c^{(k)}(u) \right)$$

Root node features

Neighboring node features

This MLP can provide "one-hot" input feature for the next layer.

The Complete GIN Model

- **GIN's node embedding updates**
- **Given:** A graph G with a set of nodes V.
  - Assign an **initial vector** $c^{(0)}(v)$ to each node v.
  - Iteratively update node vectors by

$$c^{(k+1)}(v) = \text{GINConv}\left(\left\{c^{(k)}(v), \left\{c^{(k)}(u)\right\}_{u \in N(v)}\right\}\right),$$

  Differentiable color HASH function

  where GINConv maps different inputs to different embeddings.
  - After K steps of GIN iterations, $c^{(K)}(v)$ summarizes the structure of K-hop neighborhood.

GIN and WL Graph Kernel

- **GIN can be understood as differentiable neural version of the WL graph Kernel:**

|	Update target	Update function
WL Graph Kernel	Node colors (one-hot)	HASH
GIN	Node embeddings (low-dim vectors)	GINConv

- **Advantages of GIN over the WL graph kernel are:**
  - Node embeddings are **low-dimensional**; hence, they can capture the fine-grained similarity of different nodes.
  - Parameters of the update function can be **learned for the downstream tasks.**

Expressive Power of GIN

- **Because of the relation between GIN and the WL graph kernel, their expressive is exactly the same.**
  - If two graphs can be distinguished by GIN, they can be also distinguished by the WL kernel, and vice versa.
- **How powerful is this?**
  - WL kernel has been both theoretically and empirically shown to distinguish most of the real-world graphs [Cai et al. 1992].
  - Hence, GIN is also powerful enough to distinguish most of the real graphs!

Discussion: The Power of Pooling

Failure cases for mean and max pooling:

(a) Mean and Max both fail   (b) Max fails   (c) Mean and Max both fail
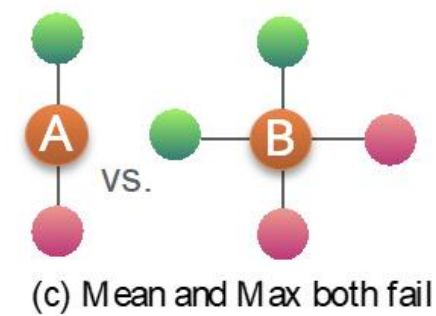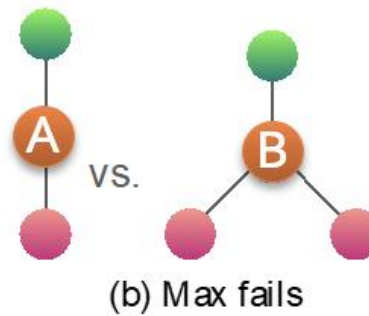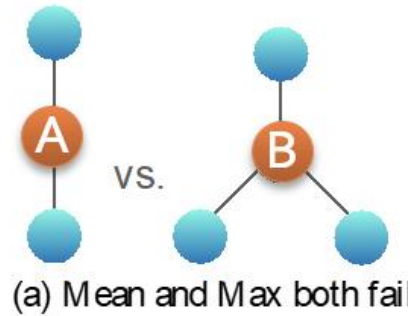
Colors represent feature values

Ranking by discriminative power:

Input   sum - multiset > mean - distribution > max - set
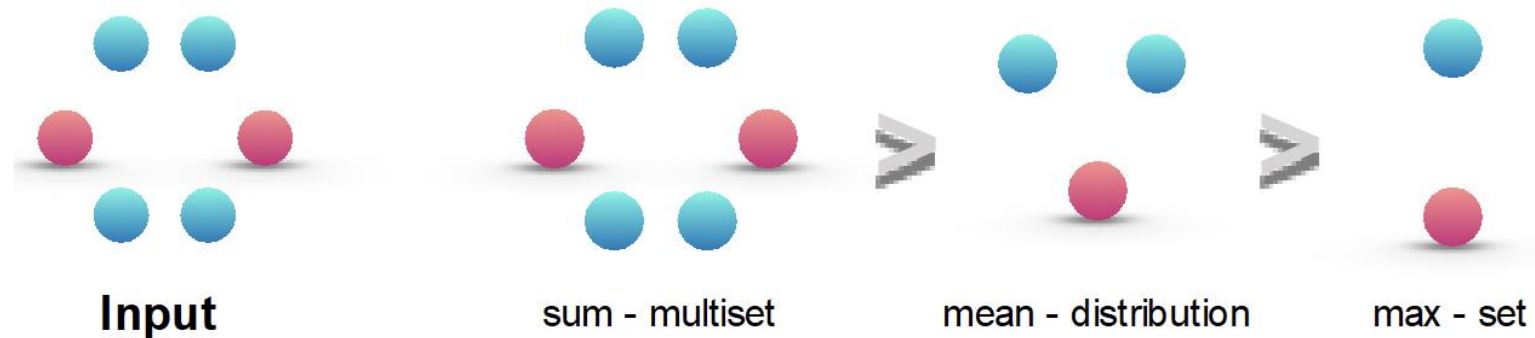
Improving GNNs' Power

- **Can the expressive power of GNNs be improved?**

  - There are basic graph structures that existing GNN framework cannot distinguish, such as difference in cycles.

Graphs

Computational graphs for nodes v_1 and v_2:

  - GNNs' expressive power **can be improved** to resolve the above problem.
    [You et al. AAAI 2021, Li et al. NeurIPS 2020]

Summary of the Lecture
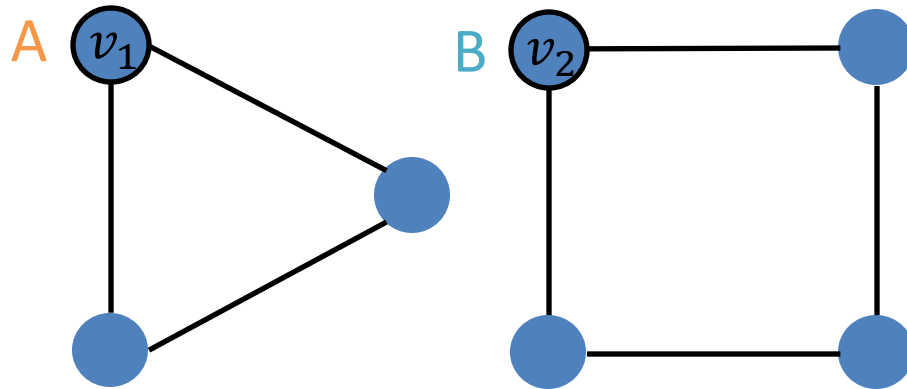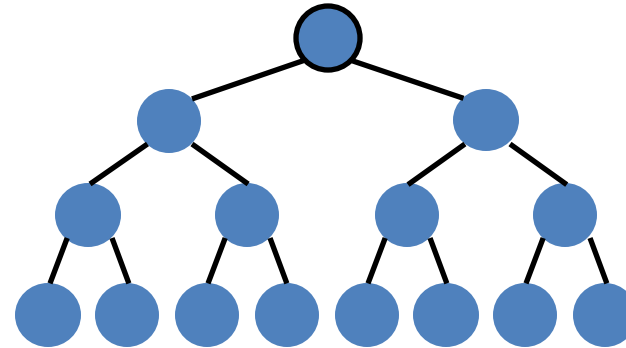
- We design a neural network that can model injective multi-set function.

- We use the neural network for neighbor aggregation function and arrive at **GIN**---the most expressive GNN model.

- The key is to use **element-wise sum pooling**, instead of mean-/max-pooling.

- GIN is closely related to the WL graph kernel.

- Both GIN and WL graph kernel can distinguish most of the real graphs!

Graph Neural Networks: Theory

When Things Don't Go As Planned

General Tips

- **Data preprocessing is important:**
  - Node attributes can vary a lot! Use **normalization**
    - E.g. probability ranges $(0,1)$, but some inputs could have much larger range, say $(-1000, 1000)$
- **Optimizer:** ADAM is relatively robust to learning rate
- **Activation function**
  - ReLU activation function often works well
  - Other good alternatives: LeakyReLU, PReLU
  - No activation function at your output layer
  - Include bias term in every layer
- **Embedding dimensions:**
 - 32, 64 and 128 are often good starting points

Debugging Deep Networks

- **Debug issues:** Loss/accuracy not converging during training

    - Check pipeline (e.g. in PyTorch we need [zero_grad](#))

    - Adjust hyperparameters such as learning rate

    - Pay attention to weight parameter [initialization](#)

    - **Scrutinize loss function!**

- **Important for model development:**

    - **Overfit on (part of) training data:**

        - With a small training dataset, loss should be essentially close to 0, with an expressive neural network

    - **Monitor the training & validation loss curve**

Resources on Graph Neural Networks

GraphGym:
Easy and flexible end-to-end GNN pipeline
based on PyTorch Geometric (PyG)

PyG   **DGL**   **GraphNets**
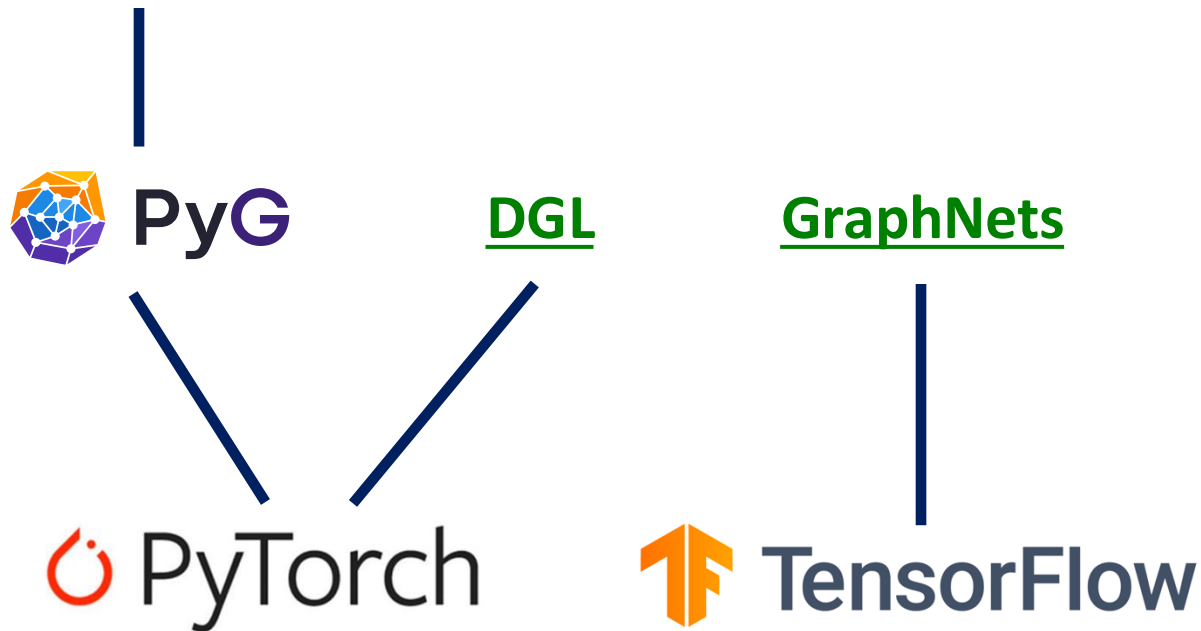
GNN frameworks:
Implements a variety
of GNN architectures

PyTorch   **TensorFlow**

Auto-differentiation frameworks

Resources on Graph Neural Networks

Tutorials and overviews:
- Relational inductive biases and graph networks (Battaglia et al., 2018)
- Representation learning on graphs: Methods and applications (Hamilton et al., 2017)

Attention-based neighborhood aggregation:
- Graph attention networks (Hoshen, 2017; Velickovic et al., 2018; Liu et al., 2018)

Embedding entire graphs:
- Graph neural nets with edge embeddings (Battaglia et al., 2016; Gilmer et. al., 2017)
- Embedding entire graphs (Duvenaud et al., 2015; Dai et al., 2016; Li et al., 2018) and graph pooling (Ying et al., 2018, Zhang et al., 2018)
- Graph generation and relational inference (You et al., 2018; Kipf et al., 2018)
- How powerful are graph neural networks(Xu et al., 2017)

Embedding nodes:
- Varying neighborhood: Jumping knowledge networks (Xu et al., 2018), GeniePath (Liu et al., 2018)
- Position-aware GNN (You et al. 2019)

Spectral approaches to graph neural networks:
- Spectral graph CNN & ChebNet (Bruna et al., 2015; Defferrard et al., 2016)
- Geometric deep learning (Bronstein et al., 2017; Monti et al., 2017)

Other GNN techniques:
- Pre-training Graph Neural Networks (Hu et al., 2019)
- GNNExplainer: Generating Explanations for Graph Neural Networks (Ying et al., 2019)