Yale

Theory and Expressive Power of Graph Neural Networks

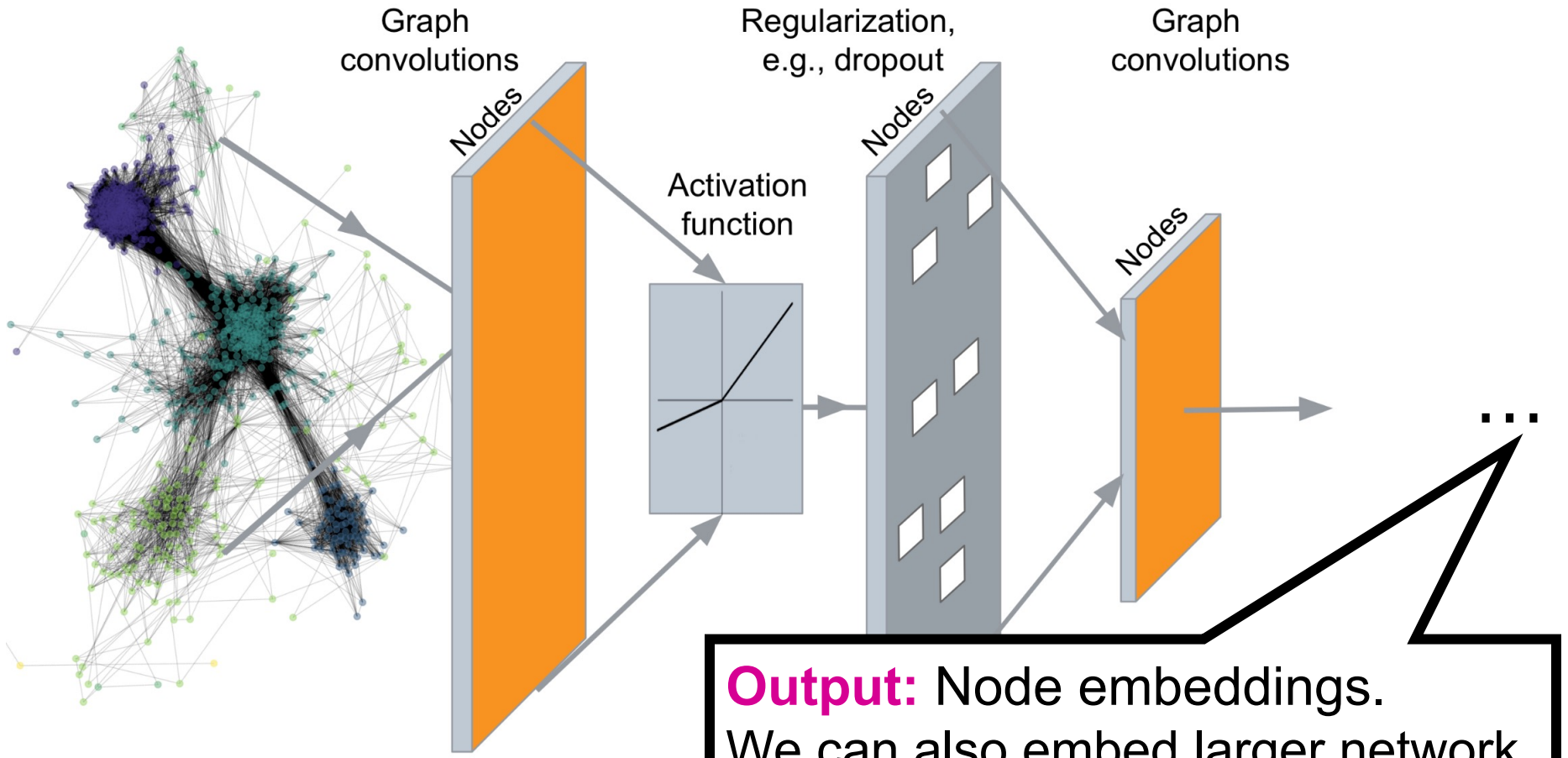
CPSC483: Deep Learning on Graph-Structured Data

Rex Ying

Readings

- Readings are updated on the website (syllabus page)
- **Lecture 8 readings:**
  - Attention is All You Need
  - Graph Structure of Neural Networks
- **Lecture 9 readings:**
  - Graph Isomorphism Network
  - Weisfeiler and Leman Go Neural: Higher-order Graph Neural Networks

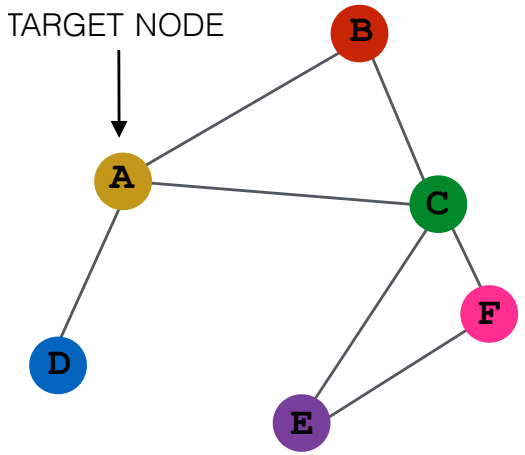
Recap: Graph Neural Networks



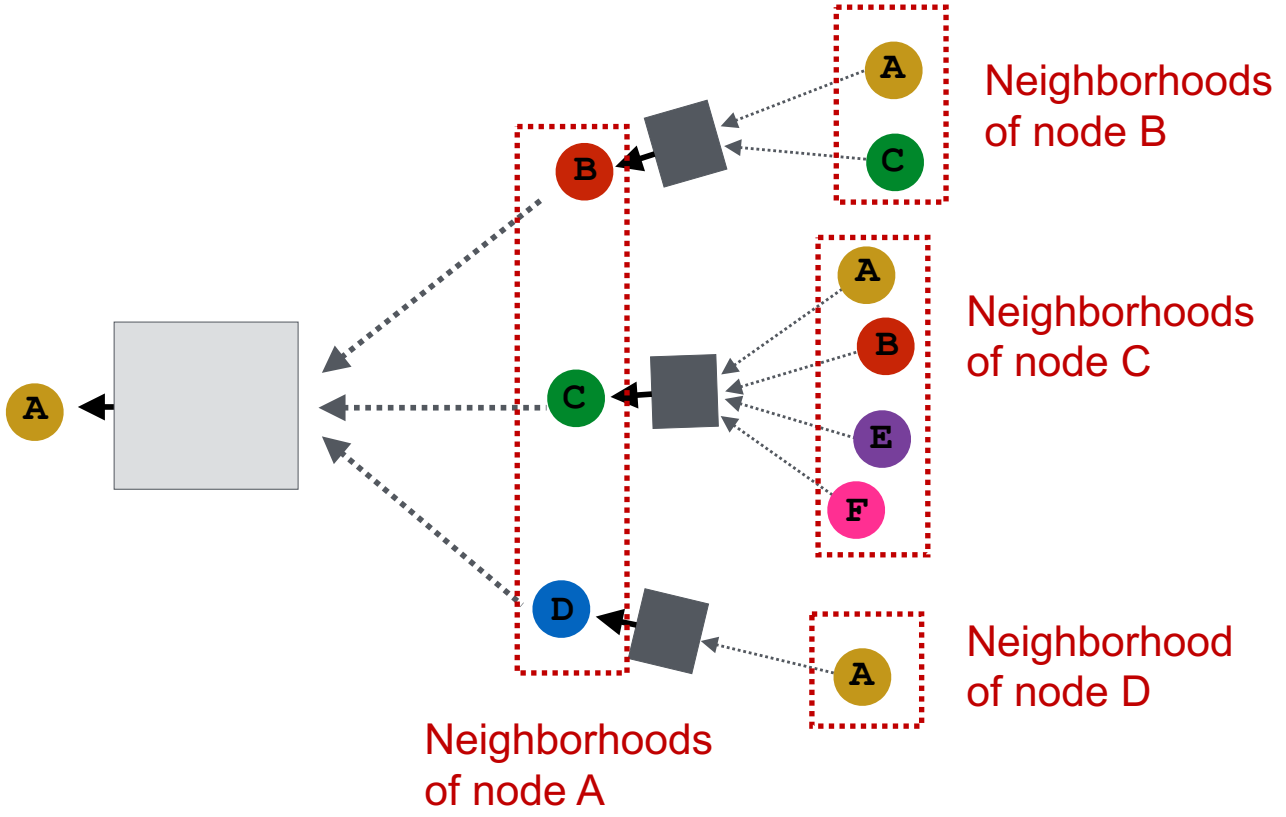
Output: Node embeddings.
We can also embed larger network structures, subgraphs, graphs.

Recap: Neighborhood Aggregation (1)

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

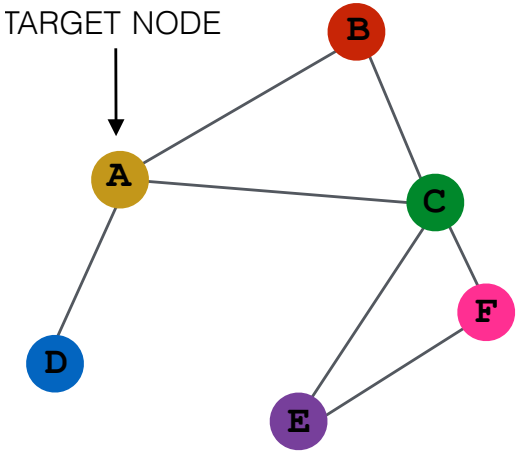


INPUT GRAPH

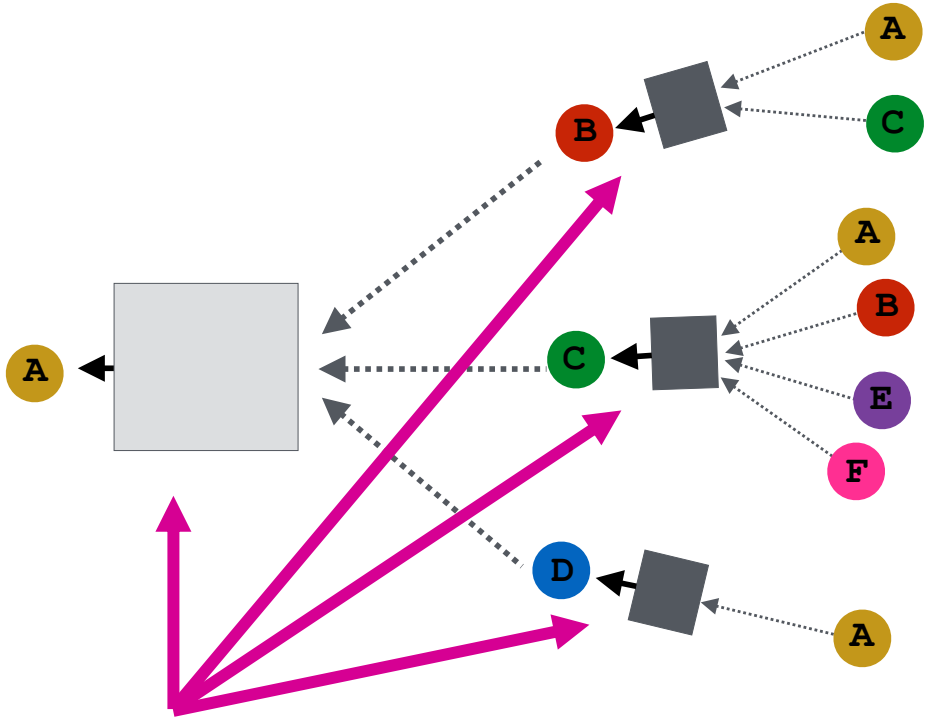


Recap: Neighborhood Aggregation (2)

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



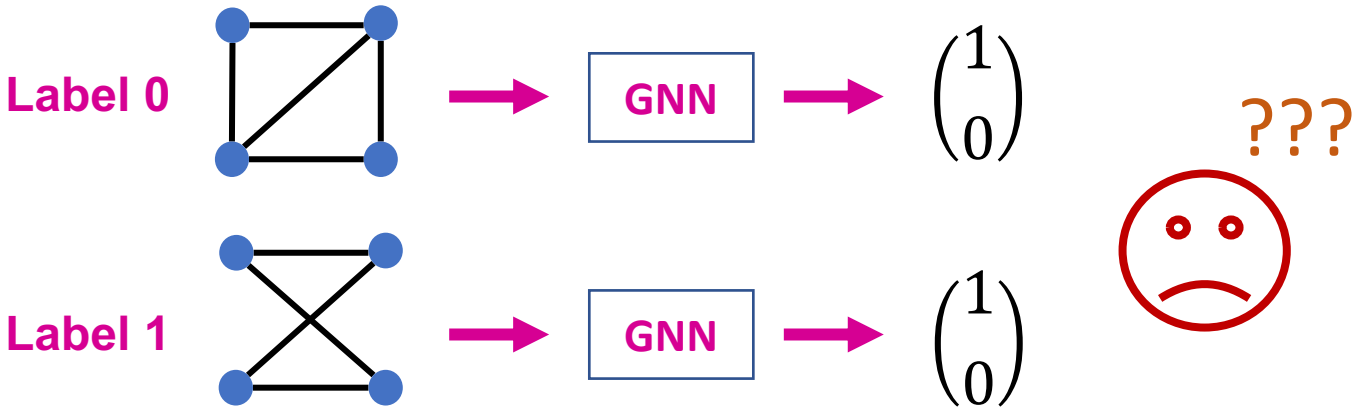
INPUT GRAPH



Neural networks

Importance of Expressive Power

- **Expressive Power** of GNNs is the ability to distinguish different graph structures.
- It is **crucial** for GNNs!
  - **Ex:** In graph classification task, if a GNN generates the **same embeddings** for two graphs **with different structures and different ground-truth labels**, it will predict the same label for them, thus **failing to distinguish them**.



Theory of GNNs

How powerful are GNNs?

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

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

How to design a maximally expressive GNN model?

Content

- **The problem of GNN Expressiveness**
- **Designing Maximally Powerful GNNs**

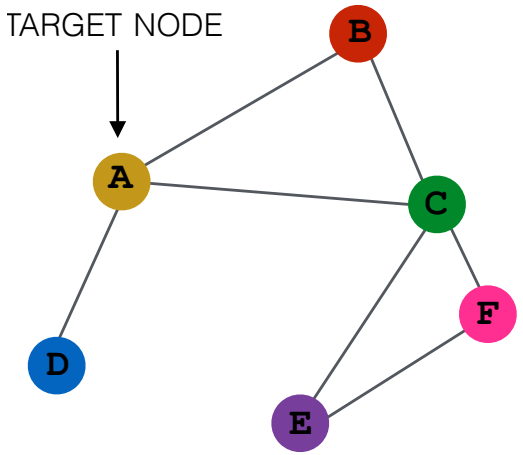
Content

- **The problem of GNN Expressiveness**
  - Computation Graph
  - Expressive Power of GNNs

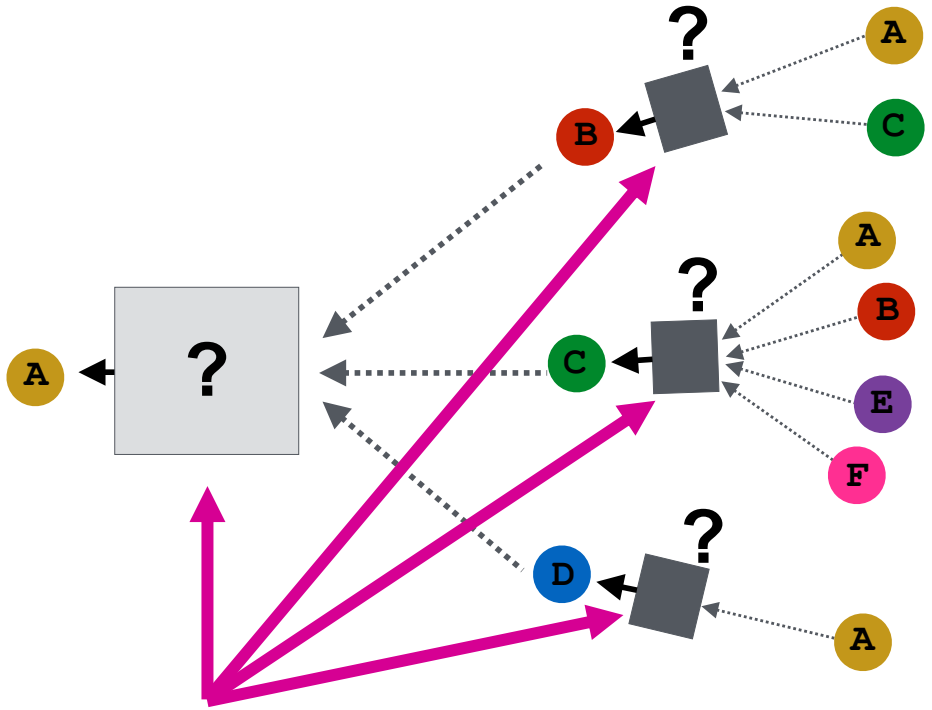
- **Designing Maximally Powerful GNNs**

Background: Many GNN Models

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



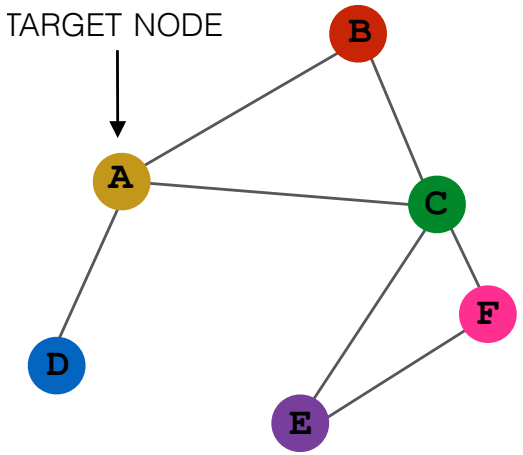
INPUT GRAPH



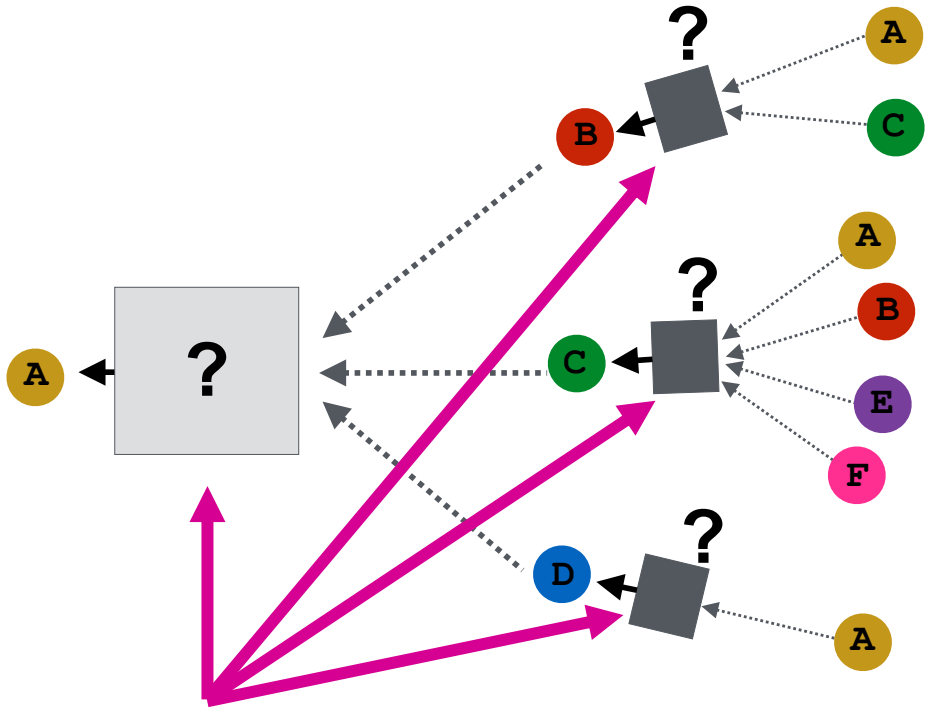
Different GNN models use different neural networks in the box

GNN Model Example (1)

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



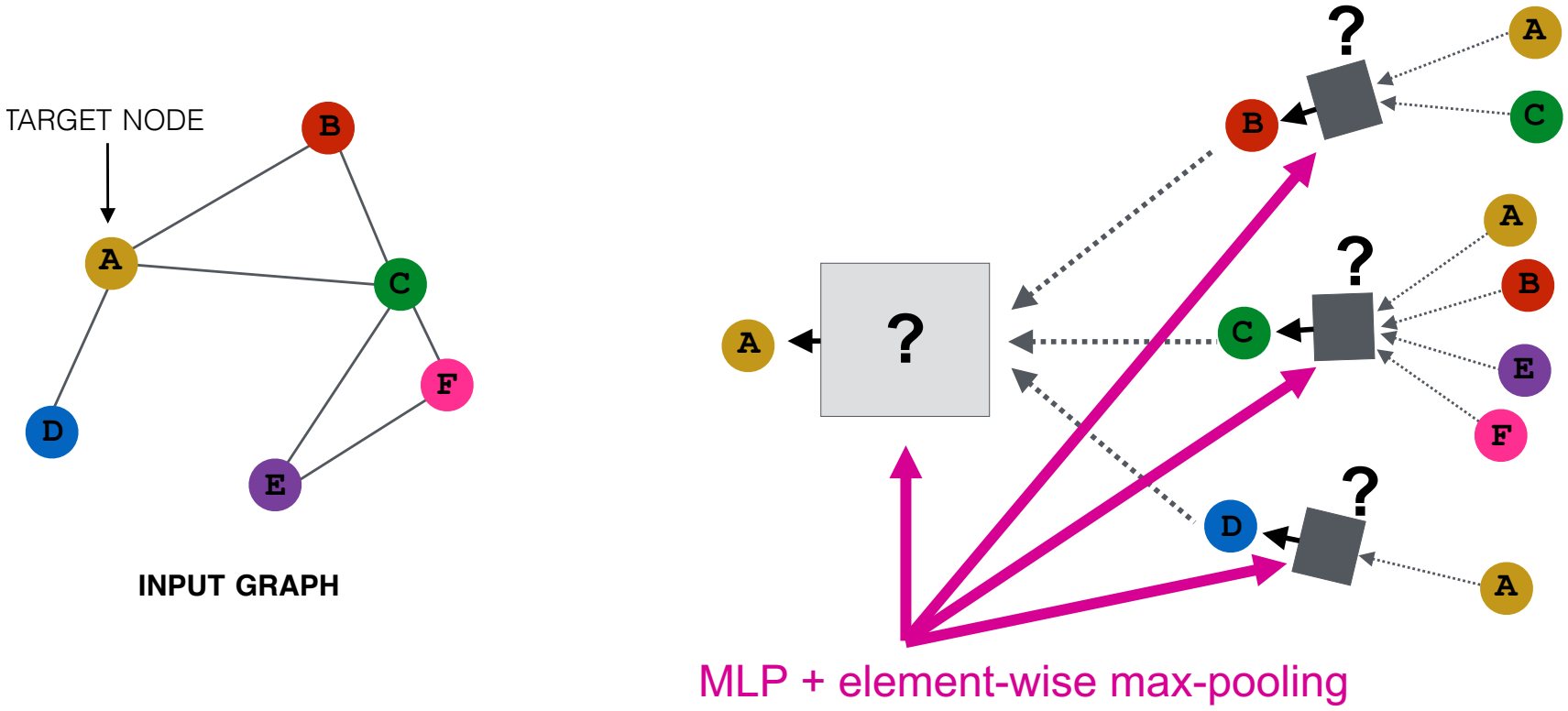
INPUT GRAPH



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

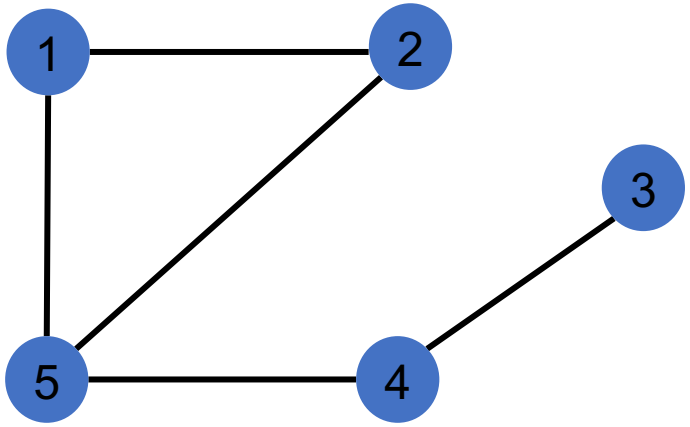
GNN Model Example (2)

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



Note: Node Colors

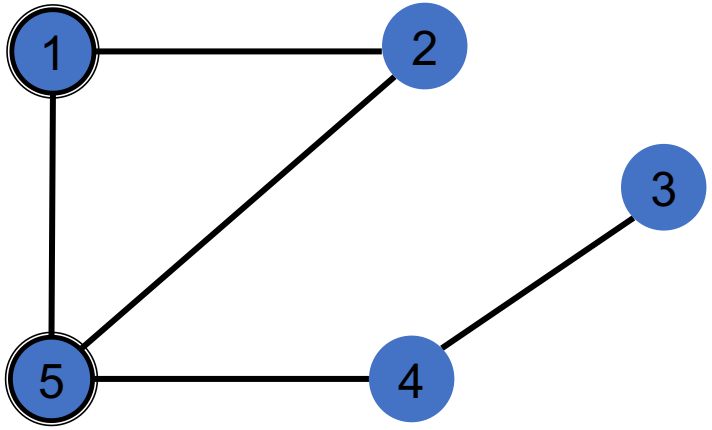
- We use node same/different **colors** to represent nodes with same/different **features**.
  - For example, the graph below assumes all the nodes **share the same features**.



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

Local Neighborhood Structures

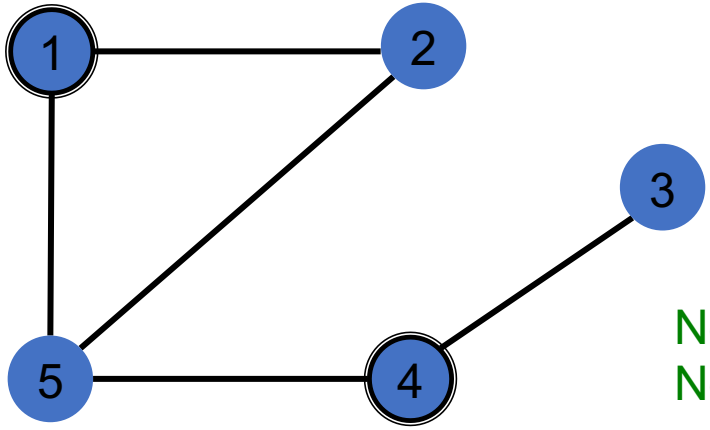
- We specifically consider **local neighborhood structures** around each node in a graph.
  - **Example:** Nodes 1 and Node 5 have **different** neighborhood structures (obviously) because they have different node degrees.



Node 1's node degree is 2
Node 5's node degree is 3

Local Neighborhood Structures

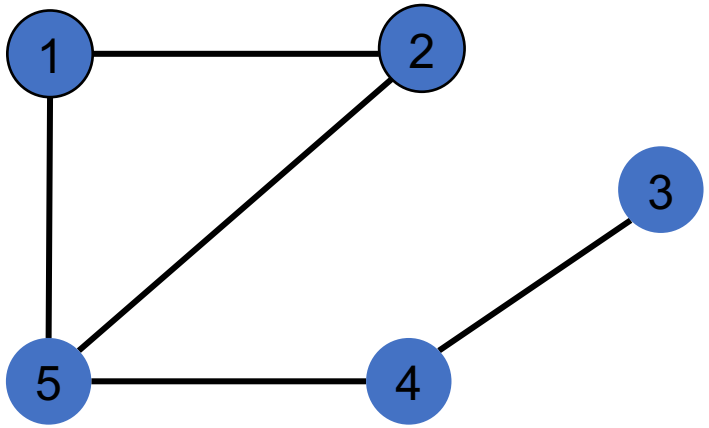
- We specifically consider **local neighborhood structures** around each node in a graph.
  - **Example:** Nodes 1 and Node 4 both have the same node degree of 2.
  - However, they still have **different** neighborhood structures because **their neighbors have different node degrees.**



Node 1 has neighbors of degrees 2 (Node 2) and 3 (Node 5).
Node 4 has neighbors of degrees 1 (Node 3) and 3 (Node 5).

Local Neighborhood Structures

- We specifically consider **local neighborhood structures** around each node in a graph.
  - **Example:** Nodes 1 and Node 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: Computation graph**

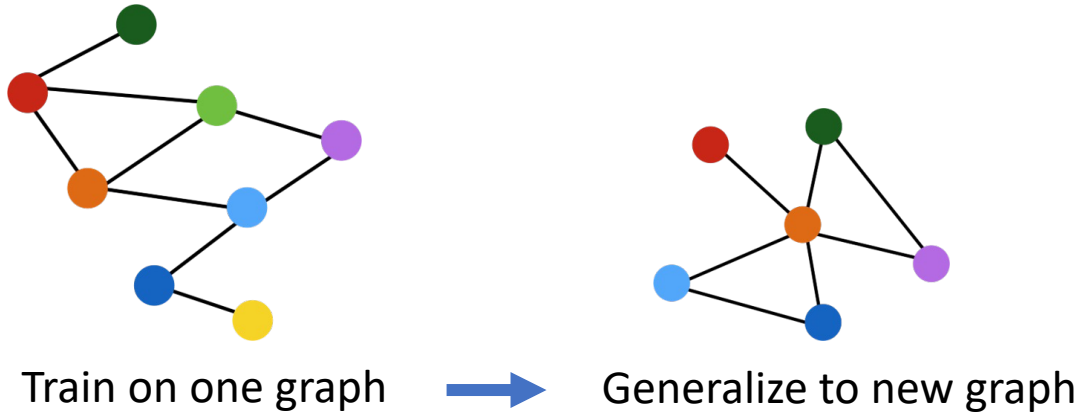
Computation Graph

- **In each layer, a GNN aggregates neighboring node embeddings.**
- A GNN generates node embeddings through a computation graph defined by the neighborhood.
- **Assumption**: we **don't consider node features and node IDs.** Only neighboring structure of the node can be used.

Computation Graph

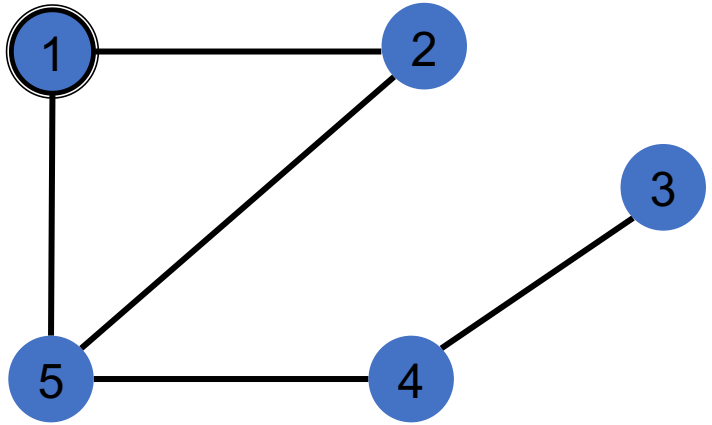
Why do we not consider node features / IDs in computation graphs?

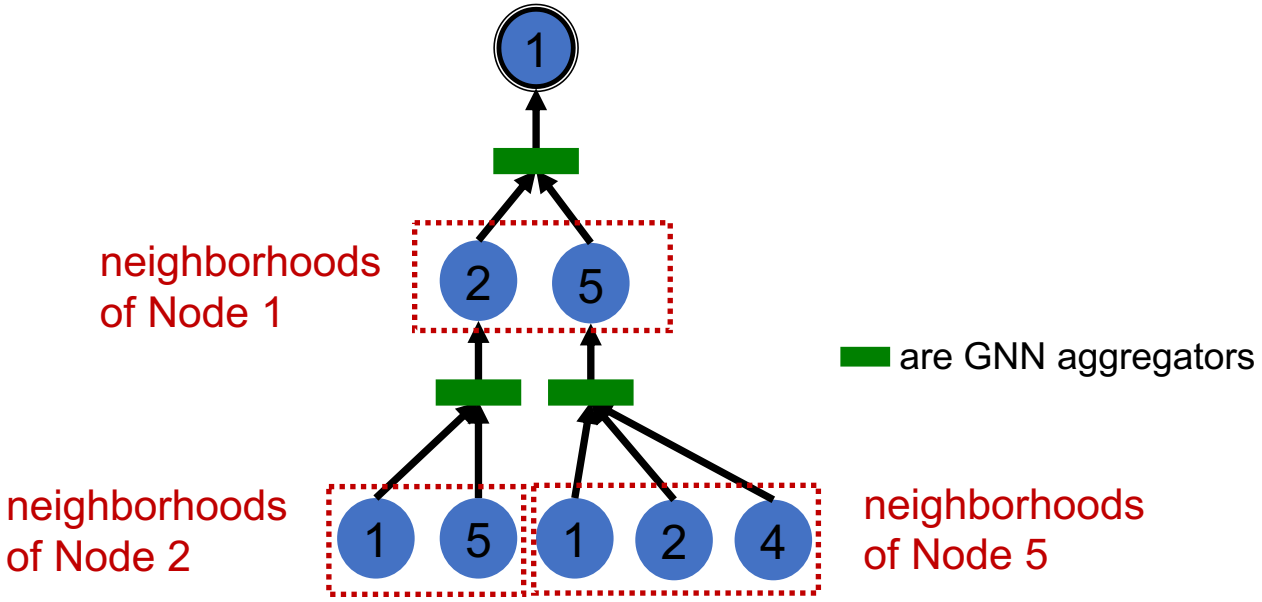
- It will be much easier to distinguish different computation graphs with features.
- We want GNNs to distinguish graph structure even when there is no features.
- For **inductive use case**, node IDs cannot be seen by GNNs.
  - **Recap:** Inductive models are able to generalize to entirely unseen graphs (lecture 5)
  - Node IDs cannot be generalized to a new graph



Computation Graph

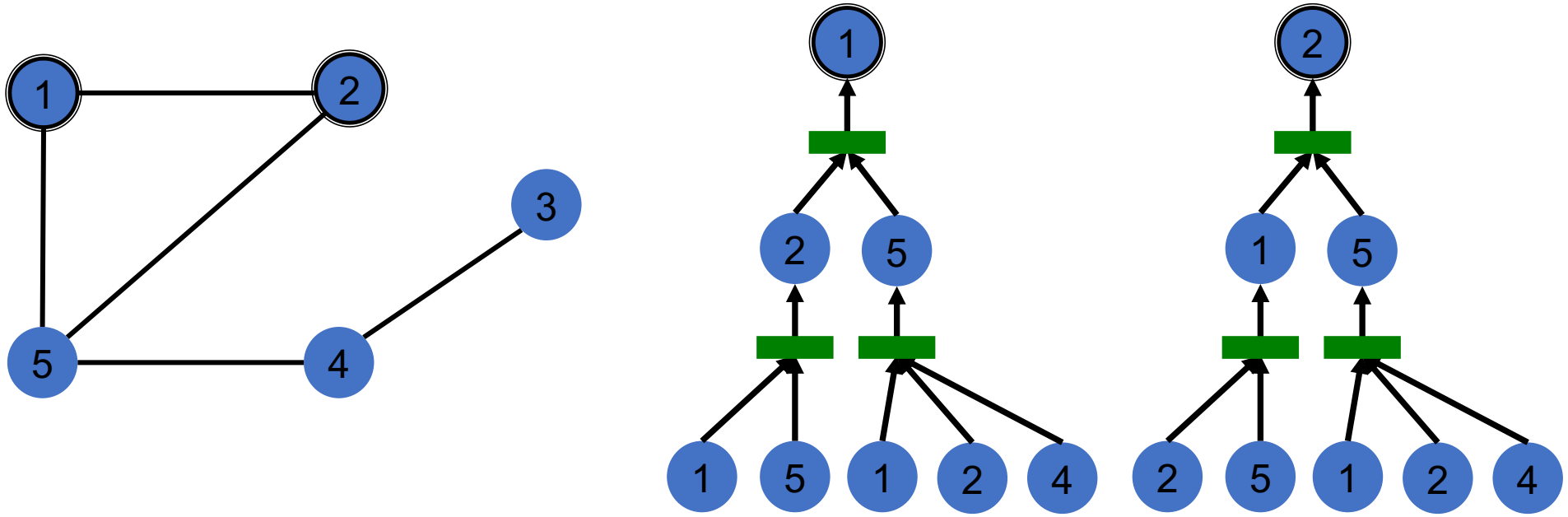
- **Example:** Node 1's computation graph (2-layer GNN)
  - A 2-layer GNN generates a computation graph using 2-hop neighborhood structure





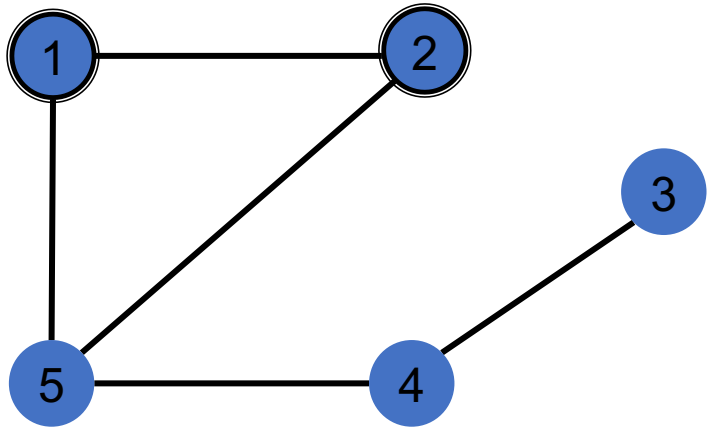
Computation Graph

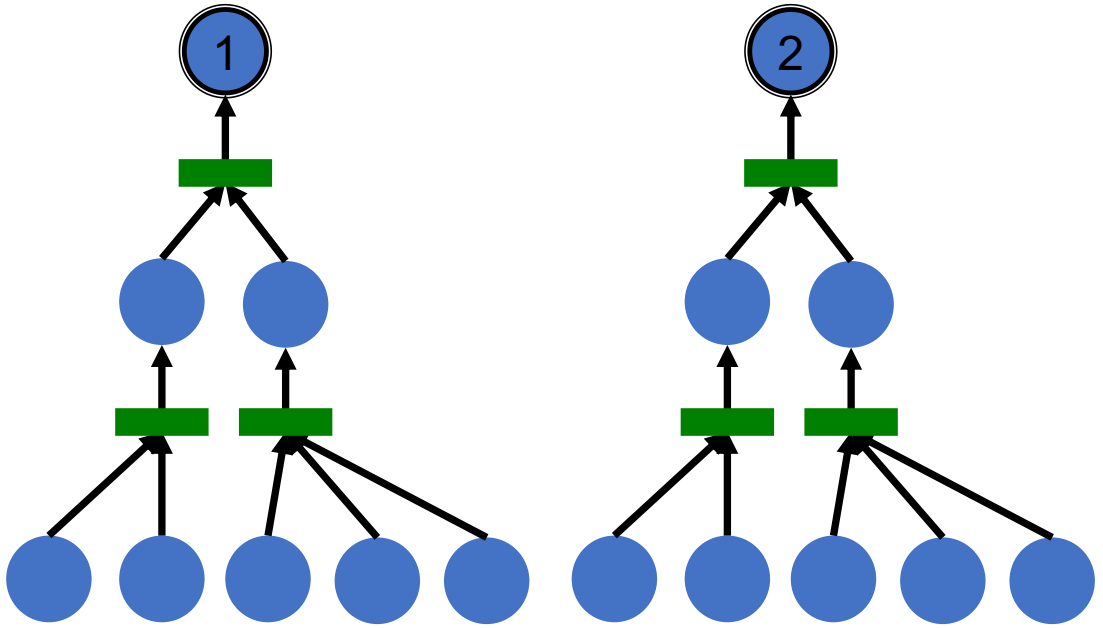
- **Example :** Nodes 1 and 2's computation graphs.



Computation Graph

- **Example:** Nodes 1 and 2's computation graphs.
- **But GNN only sees node features (not IDs):**

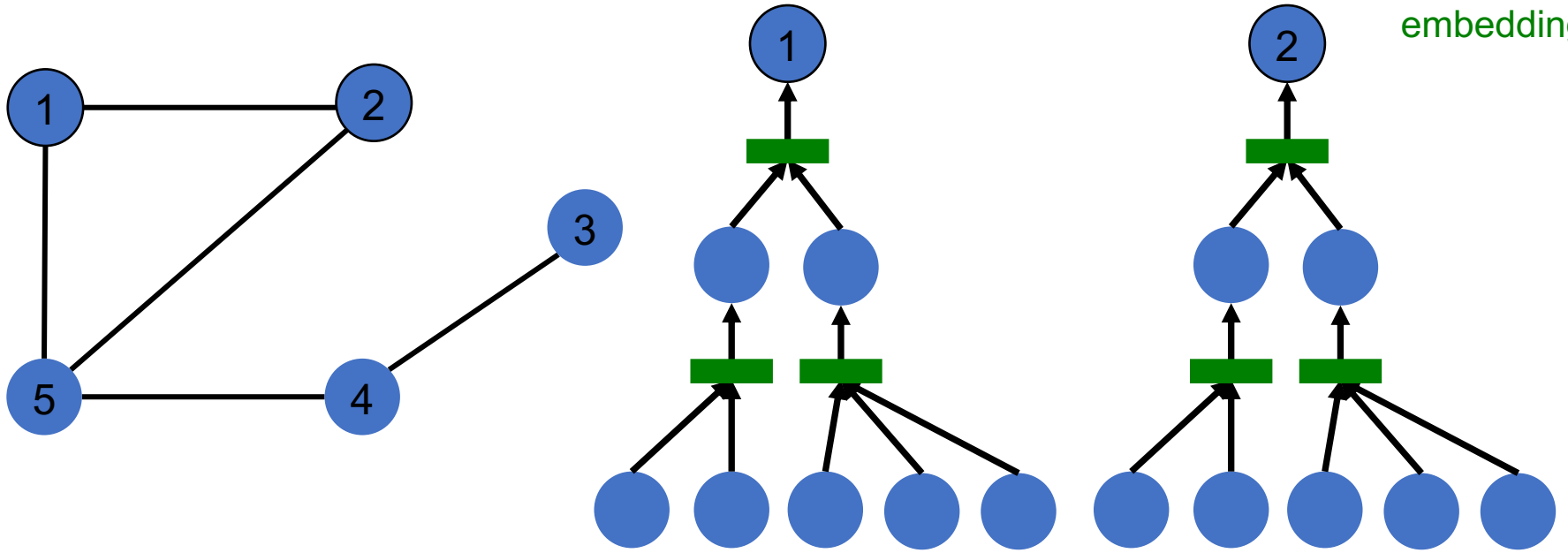




Computation Graph

- **A GNN will generate the same embedding for nodes 1 and 2 because:**
  - Computation graphs are the same (no ID will be seen).
  - Node features (colors) are identical.

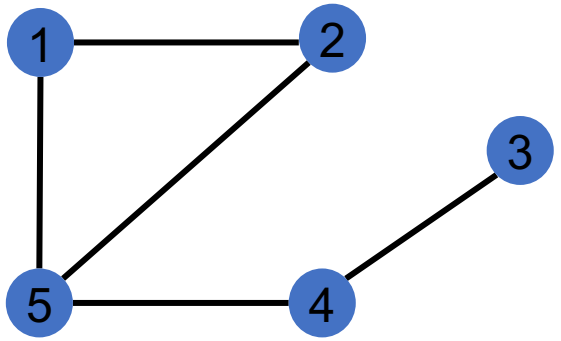
Note: GNN does not care about node ids, it just aggregates node embeddings.



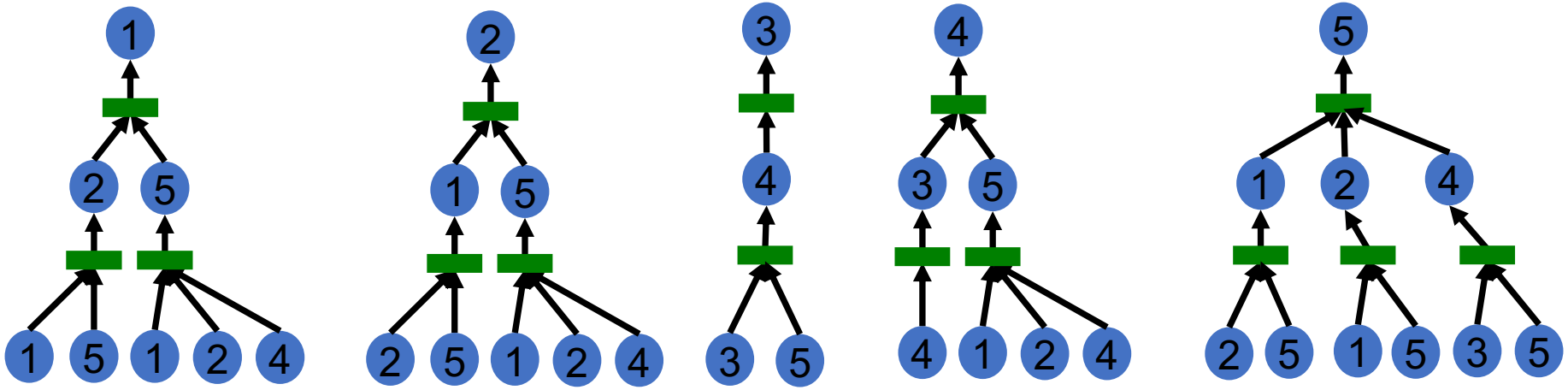
GNN won't be able to distinguish nodes 1 and 2

Computation Graph

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

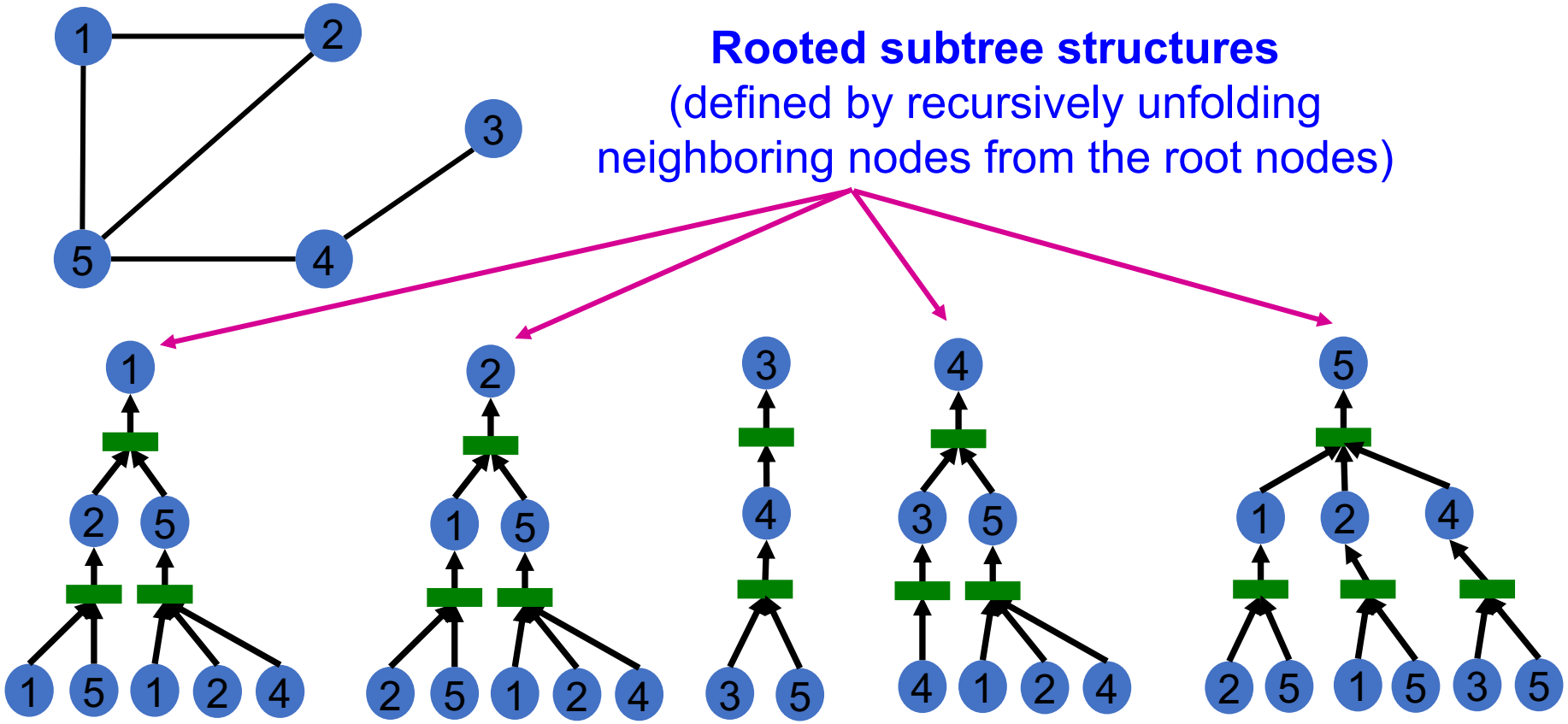


Computation graph of **Node 1 is the same as Node 2's**, but different from Node 3's, Node 4's, and Node 5's.



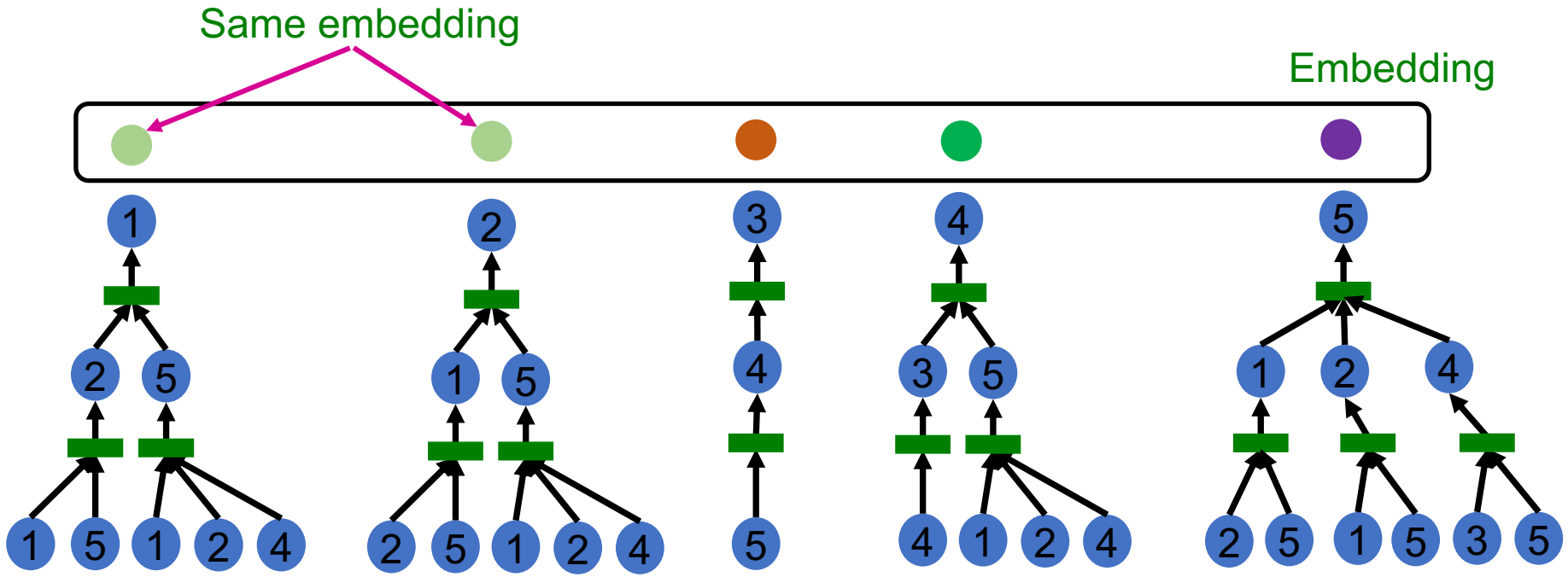
Rooted Subtree Structures

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



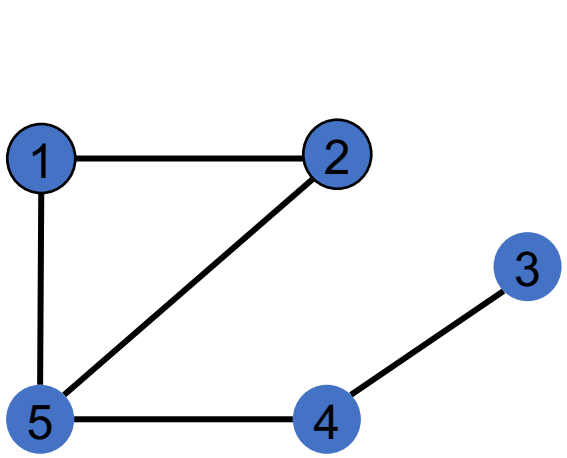
Rooted Subtree Structures

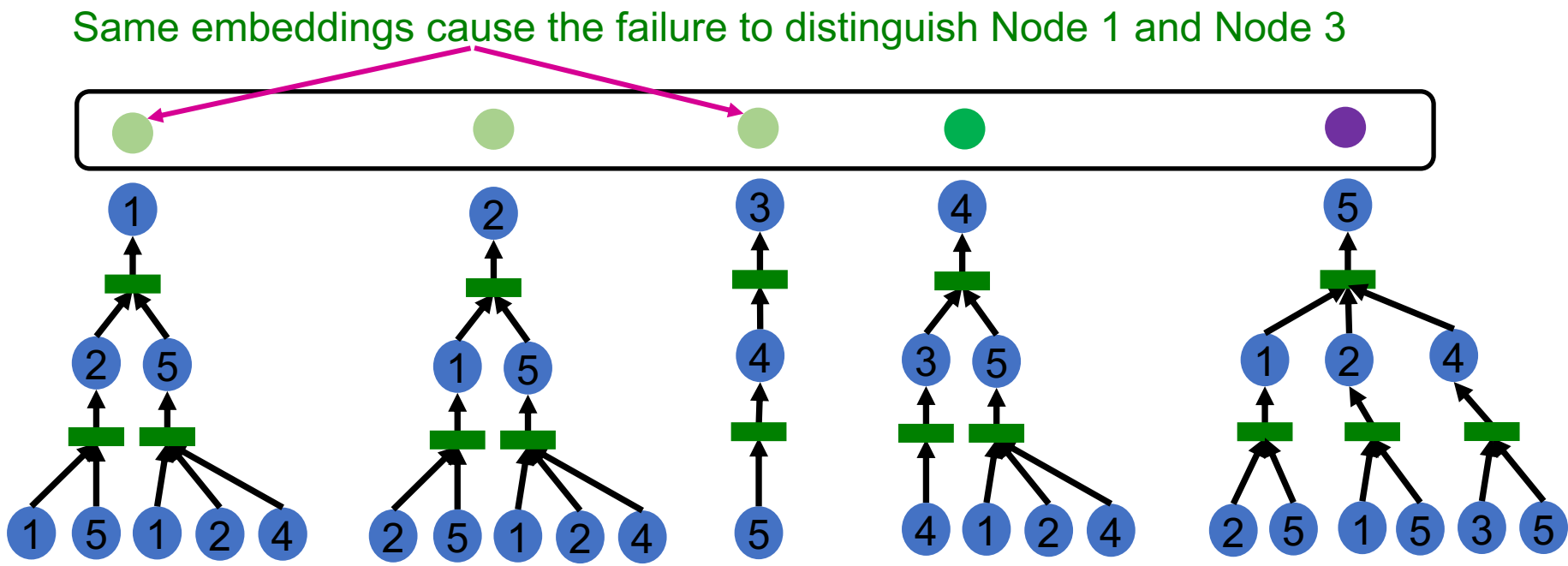
- GNN's node embeddings capture **rooted subtree structures**.
- Maximally expressive GNN should maps different rooted subtrees into different node embeddings (represented by different colors).



Not Expressive GNNs

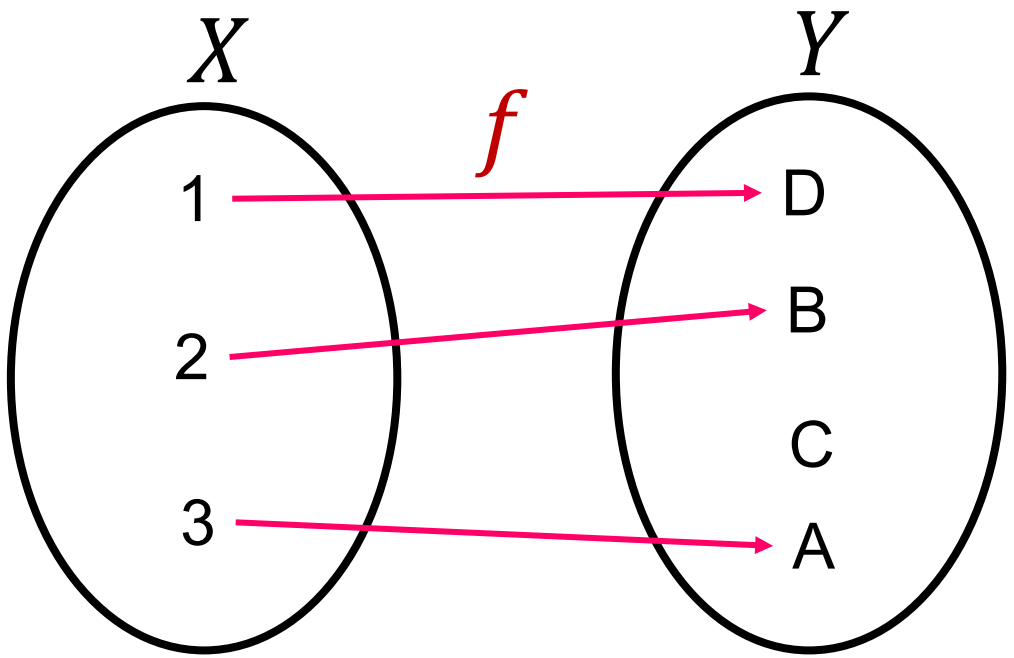
- **Question: how a maximally expressive GNN should be?**
- If a GNN maps two different rooted subtrees to the same embedding, **such a GNN is not expressive enough!**
  - **EX:** node 1 and node 3 have different neighborhoods (even different node degree)





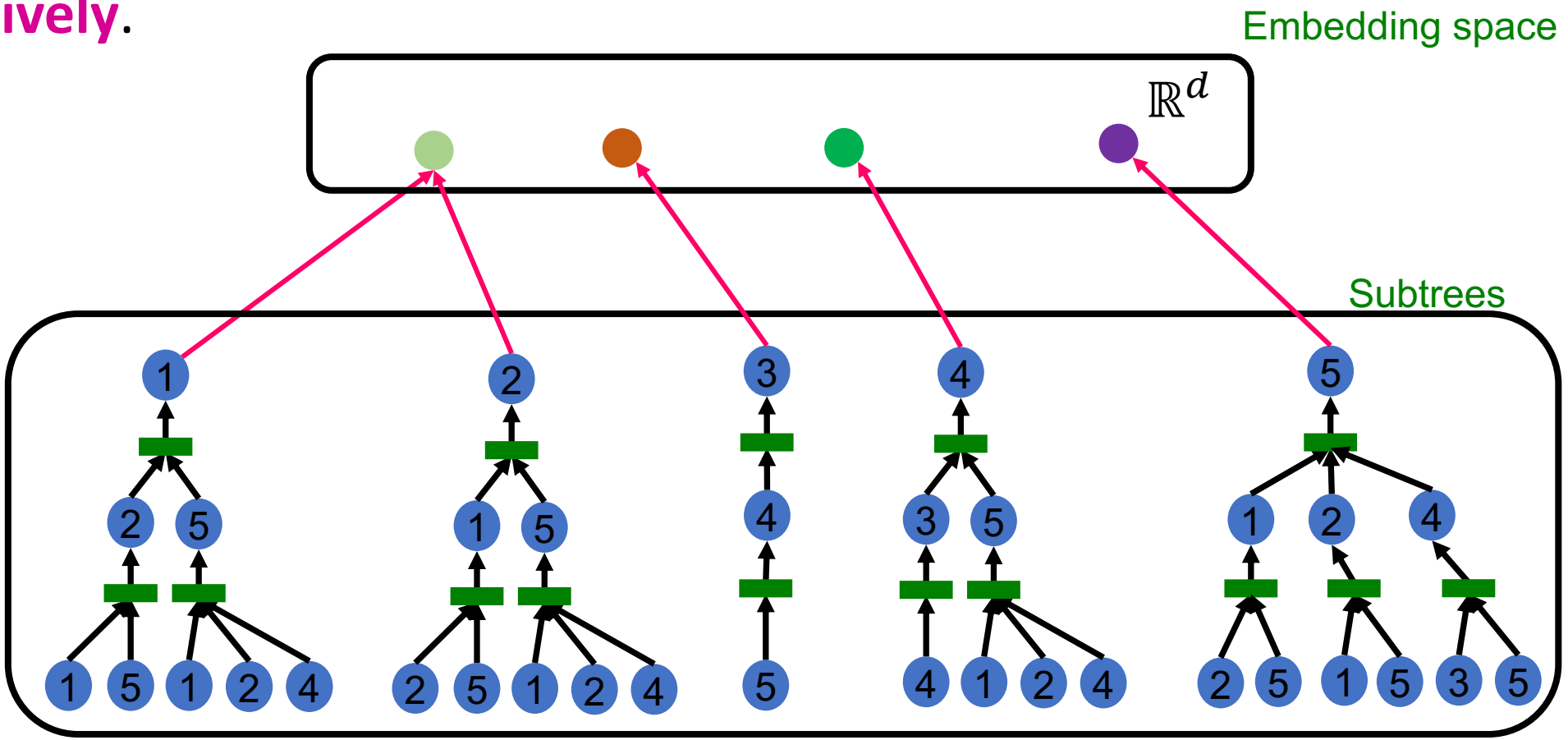
Recall: Injective Function

- **Function** $f: X \to \mathrm{Y}$ is **injective** if it maps different elements into different outputs.
- **Intuition:** f retains all the information about input.



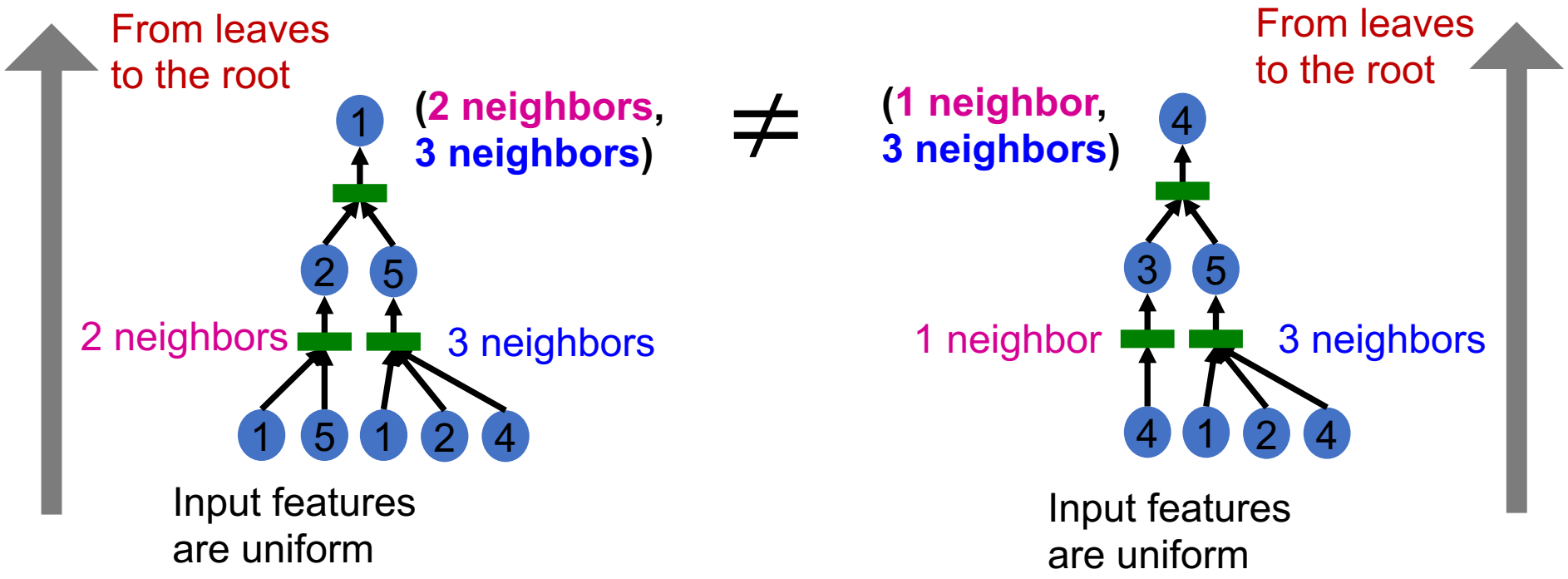
How Expressive is a GNN?

- Maximally expressive GNN should map subtrees to the node embeddings **injectively**.



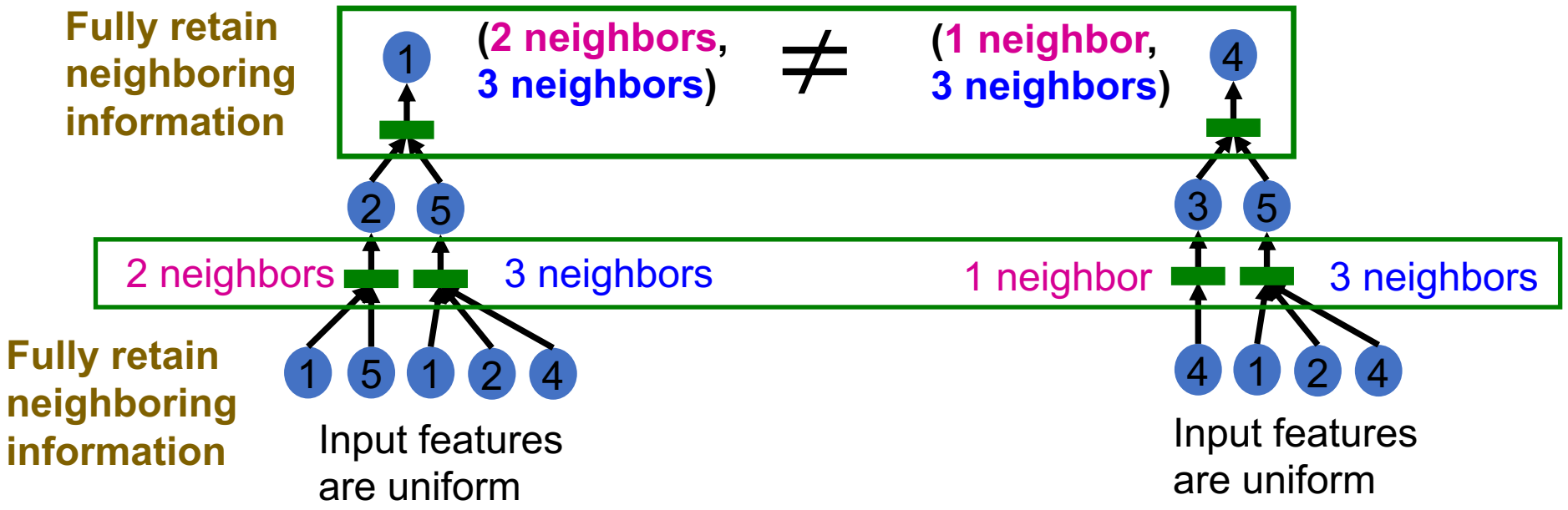
How Expressive is a GNN?

- **Key observation**: Subtrees of the same depth can be recursively characterized from the leaf nodes to the root nodes.



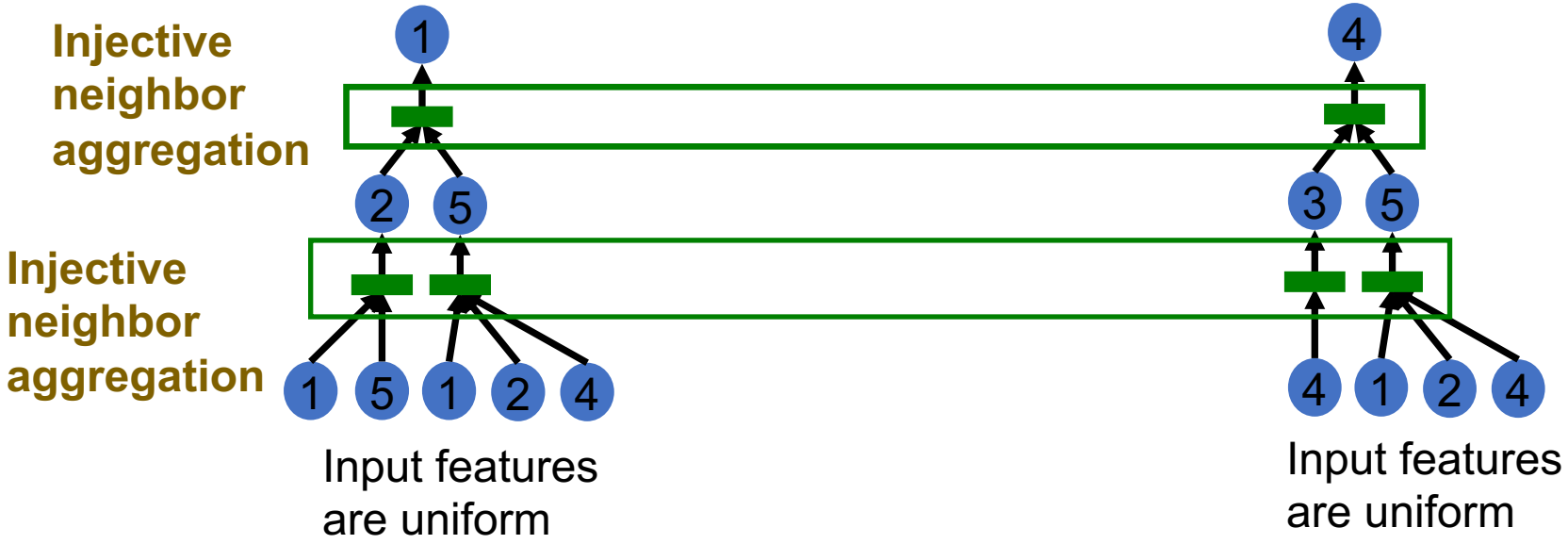
How Expressive is a GNN?

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



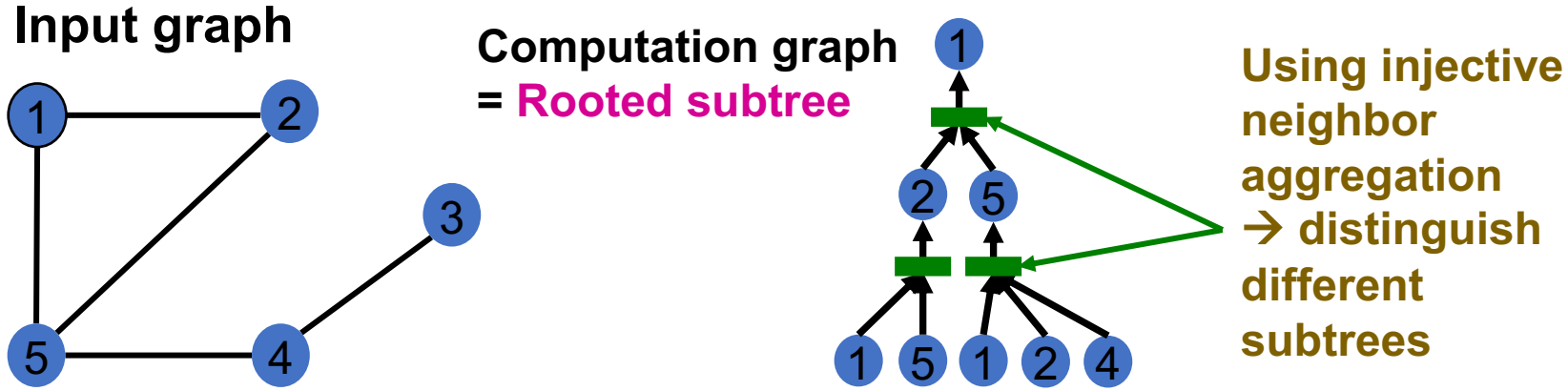
How Expressive is a GNN?

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



How Expressive is a GNN?

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



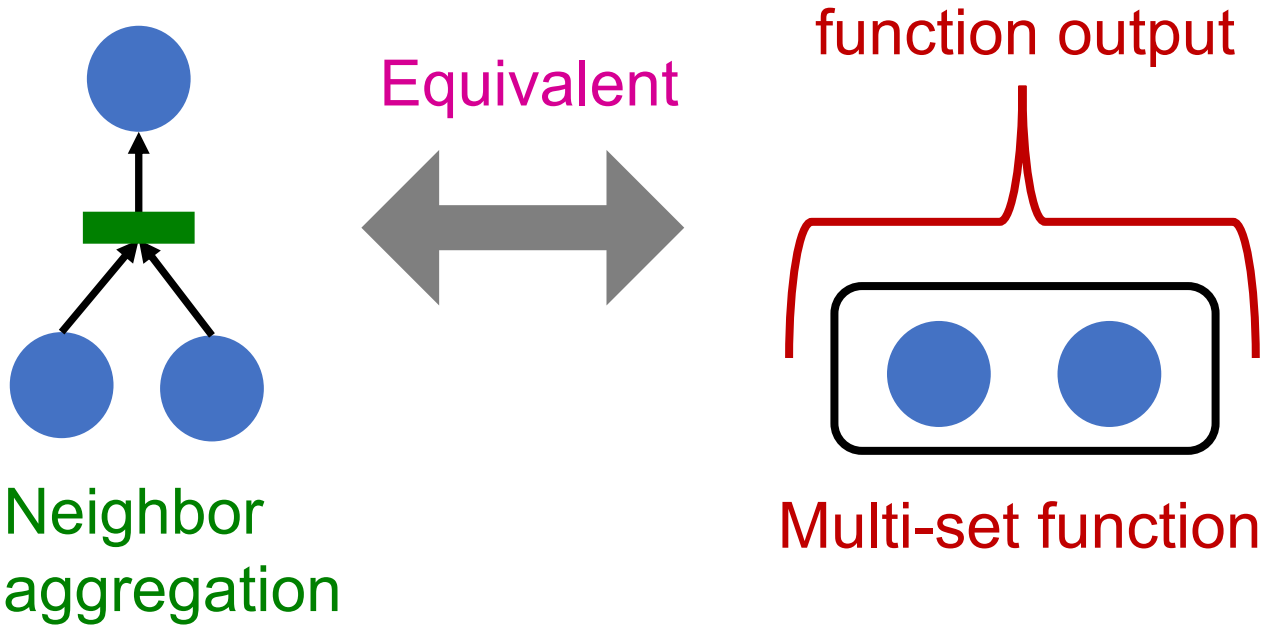
  - GNN can fully distinguish different subtree structures if **every step of its neighbor aggregation is injective.**

Expressive Power of GNNs

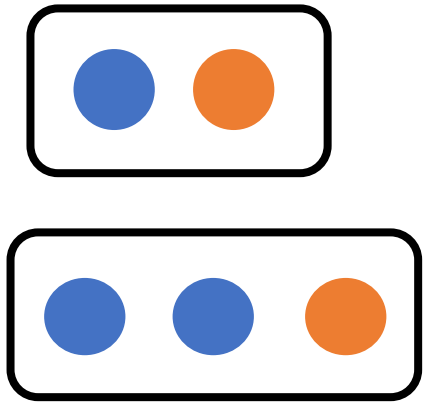
- **Key observation: Expressive power of GNNs can be characterized by that of neighbor aggregation functions they use.**
  - A more expressive aggregation function leads to a more expressive GNN.
  - Injective aggregation function leads to the most expressive GNN so far.
- **Next:**
  - Theoretically analyze expressive power of different aggregation functions.

Neighbor Aggregation

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



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

$$\text{Mean}(\{x_u\}_{u \in N(v)})$$

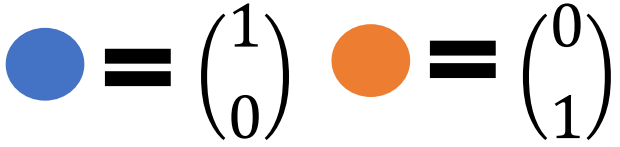
  - **GraphSAGE** (max-pool) [Hamilton et al. NeurIPS 2017]
    - Uses **element-wise** max pooling over neighboring node features

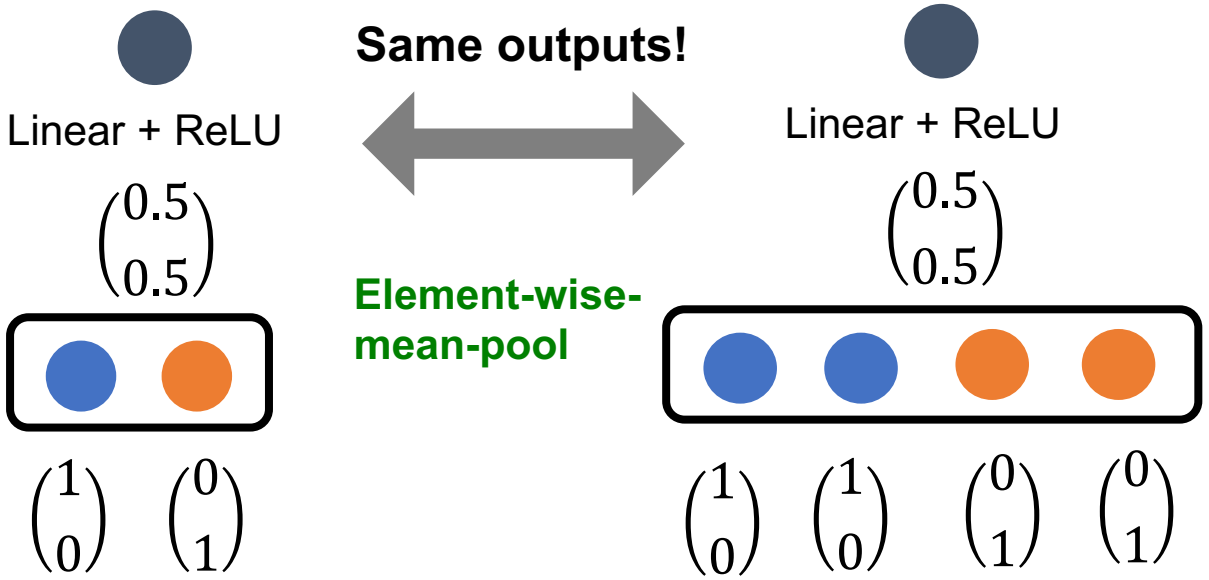
$$\text{Max}(\{x_u\}_{u \in N(v)})$$

Neighbor Aggregation: GCN

- **GCN (mean-pool)** [Kipf & Welling ICLR 2017]
  - Take **element-wise mean pool**, 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:**

If there are two distinct colors:

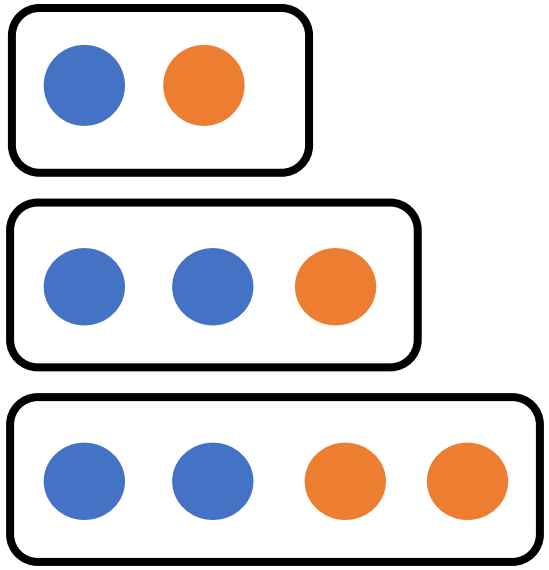




Neighbor Aggregation: GraphSAGE

- **GraphSAGE (max-pool)** [Hamilton et al. NeurIPS 2017]
  - Apply an MLP, then take **element-wise max-pool**
  - **Theorem** [Xu et al. ICLR 2019]: GraphSAGE's aggregation function cannot **distinguish different multi-sets with the same set of distinct colors**.

Failure case:
Different multi-sets but with
the same set of distinct colors



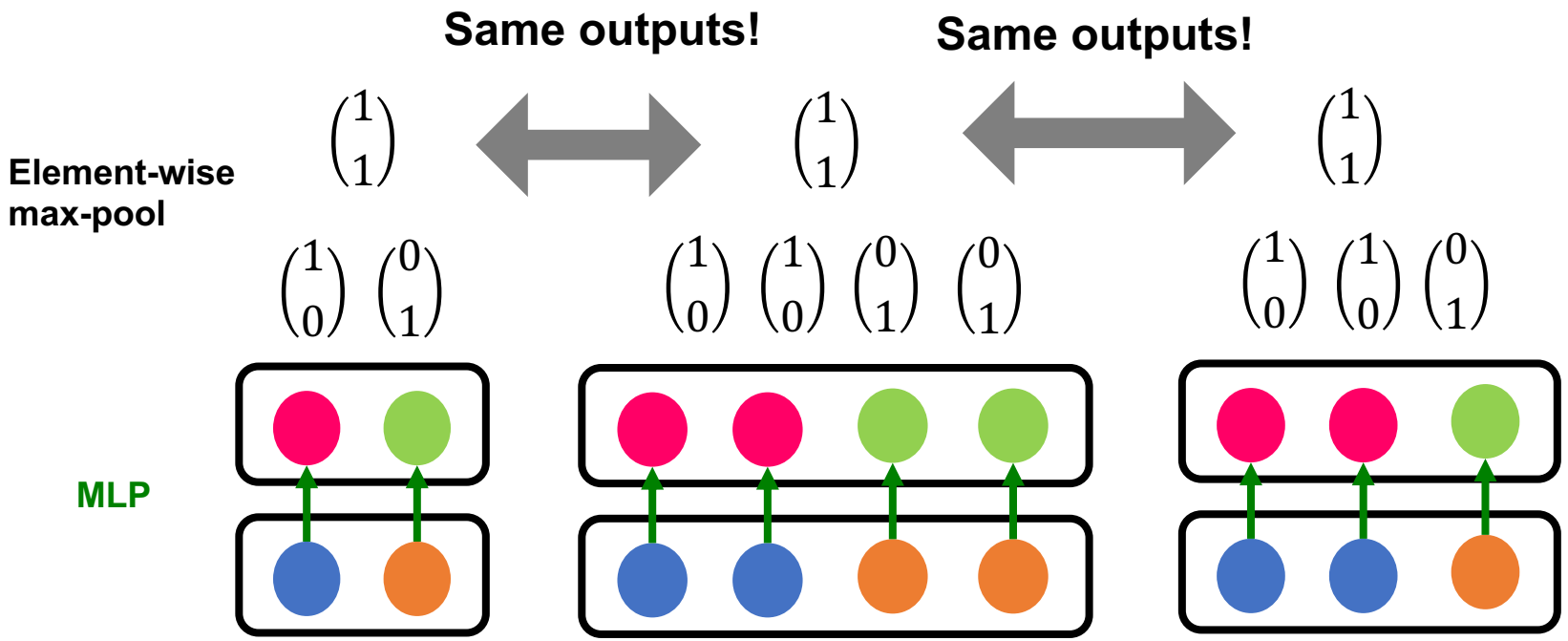
Neighbor Aggregation: GraphSAGE

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

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

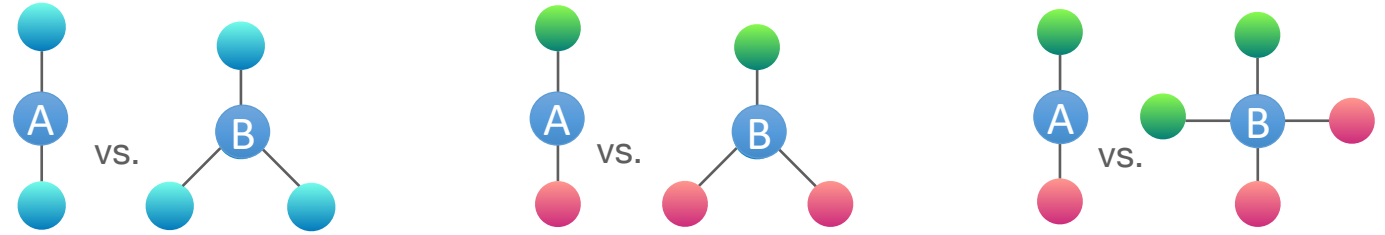
● = $\begin{pmatrix}0\\1\end{pmatrix}$

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



The Power of Pooling

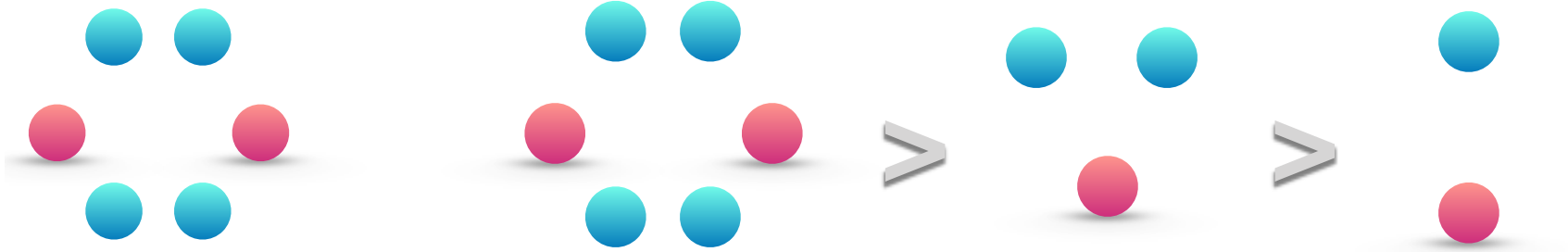
Failure cases for mean and max pooling:



(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

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.

Content

- The problem of GNN Expressiveness
  - Computation Graph
  - Expressive Power of GNNs

- Designing 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 neighborhood 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:

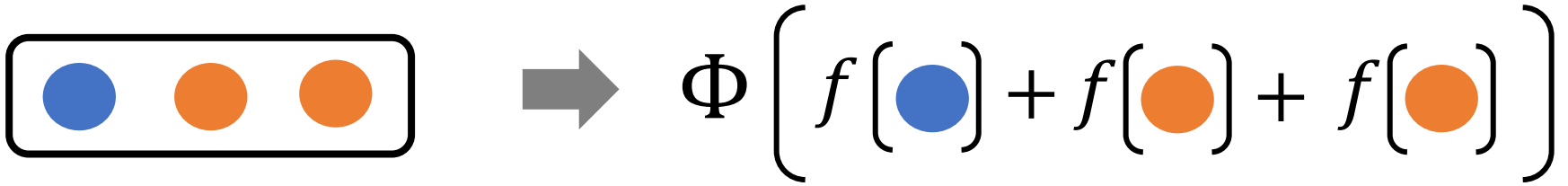
Some non-linear function

Some non-linear function

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

Sum over multi-set

S : multi-set



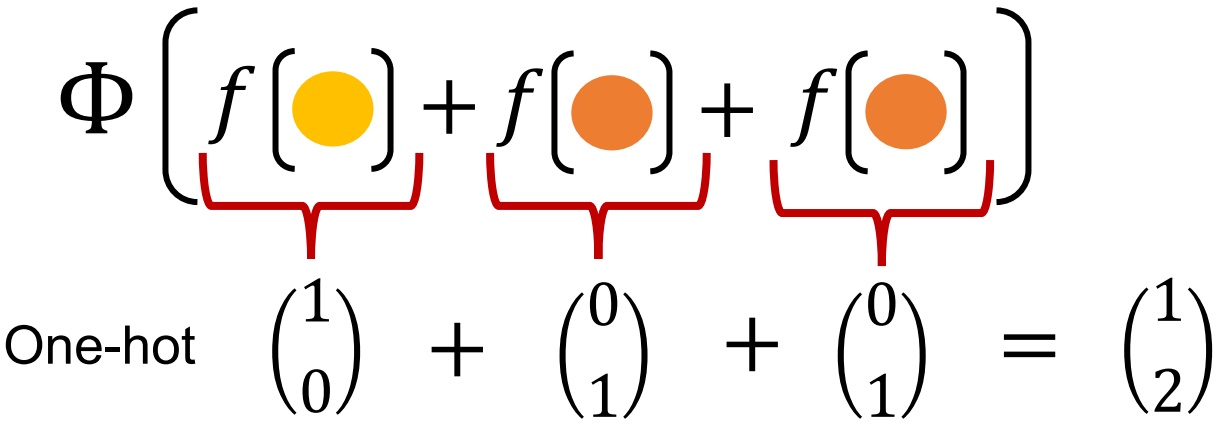
Injective Multi-Set Function

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.

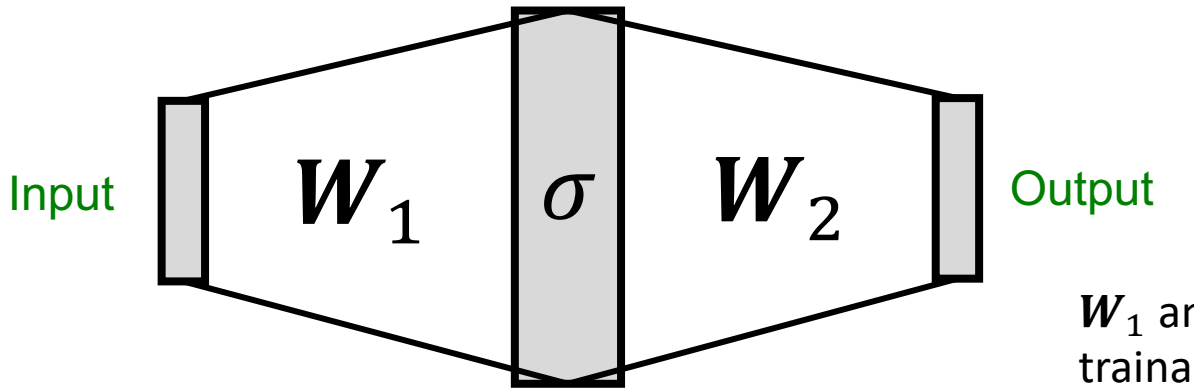
$$\Phi\left(\sum_{x \in S} f(x)\right)$$

Example:



Universal Approximation Theorem

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



W_1 and W_2 denote the trainable weights of MLP

Injective Multi-Set Function

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

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

  - 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.
- **GIN is the most expressive GNN in the class of message-passing GNNs!**

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

- **Recall: Color refinement algorithm in WL kernel.**
  - color in graph theory refers to a distinct node label
- **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

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

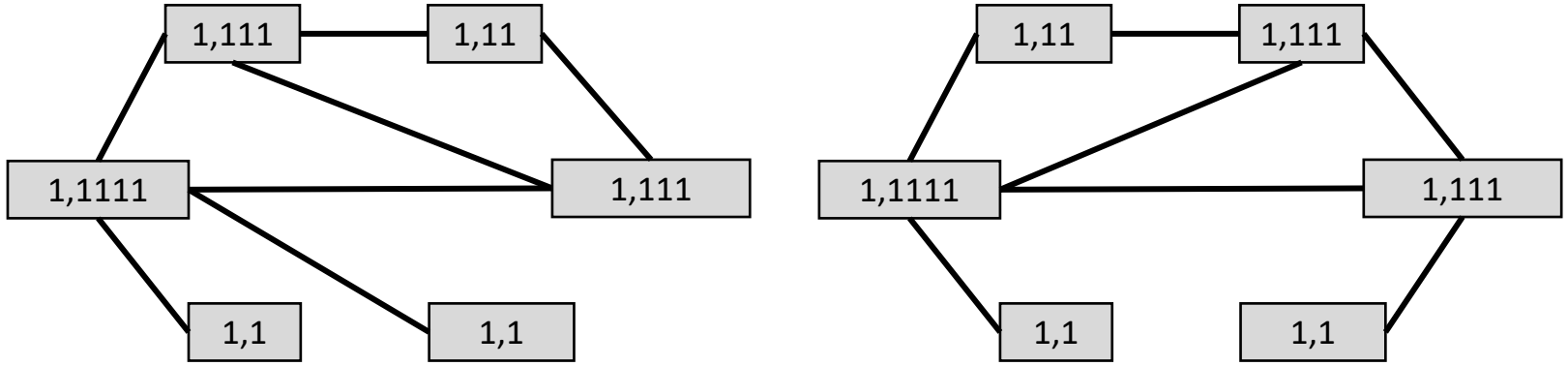
    where HASH maps different inputs to different colors.
  - After K steps of color refinement, $c^{(K)}(v)$ summarizes **the structure of K-hop** neighborhood.

Color Refinement (1)

Example of color refinement given two graphs

- Assign initial colors

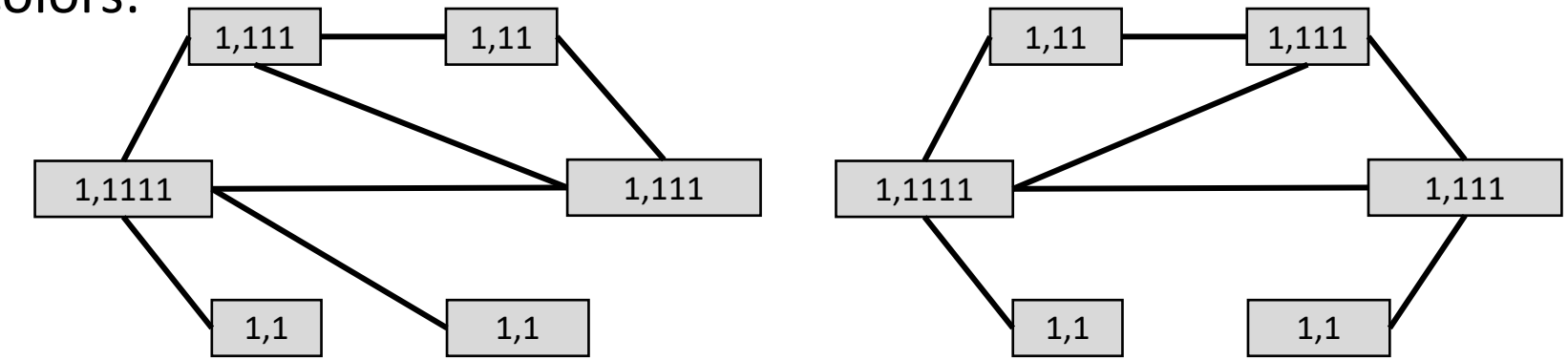
- Aggregate neighboring colors



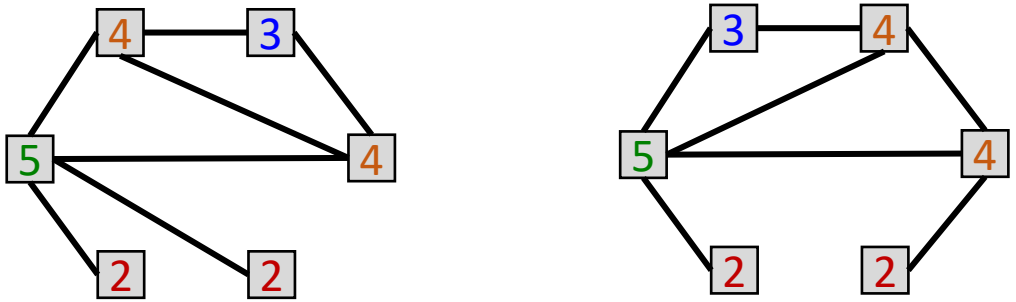
Color Refinement (2)

Example of color refinement given two graphs

- Aggregated colors:



- **Injectively** HASH the aggregated colors



HASH table: **Injective!**

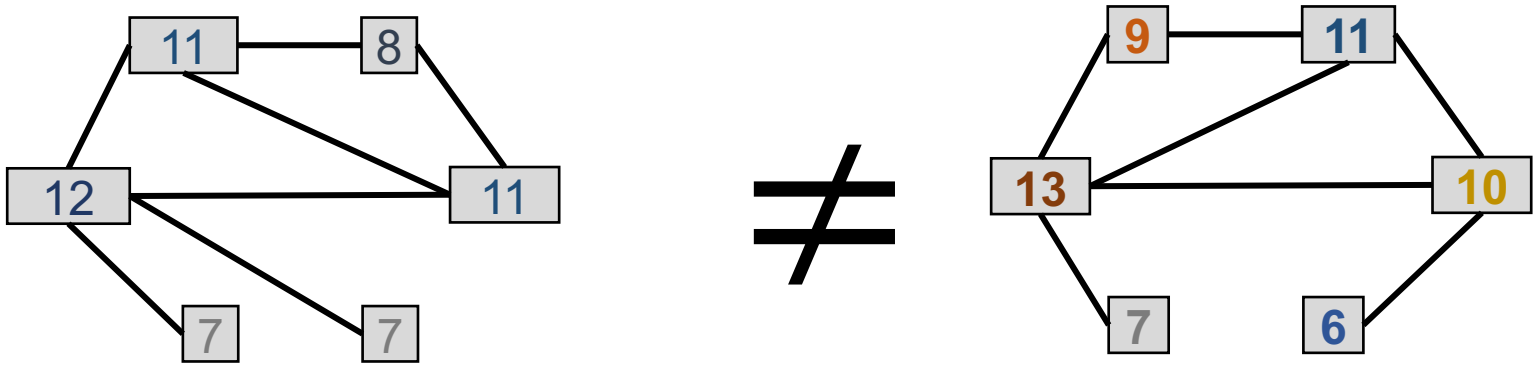
1,1	-->	2
1,11	-->	3
1,111	-->	4
1,1111	-->	5

Color Refinement (3)

Example of color refinement given two graphs

- Process continues until a stable coloring is reached

WL Isomorphic Test: Two graphs are considered **non-isomorphic** if they have different set of colors after color refinement steps.



Non-isomorphic!

The Complete GIN Model

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

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

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

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

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

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

Neighboring node features

can be modeled as

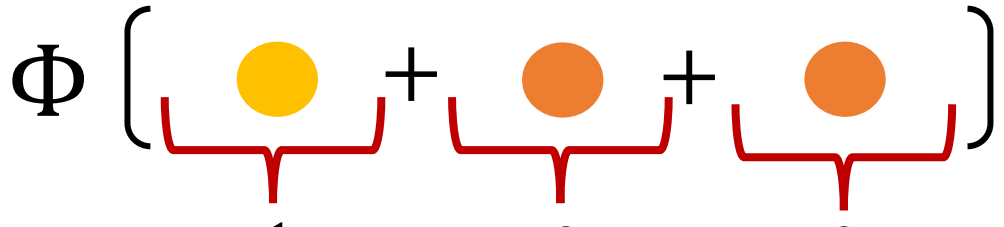
$$\mathrm{MLP}_\Phi\left((1+\epsilon)\cdot \mathrm{MLP}_f\left(c^{(k)}(v)\right) + \sum_{u \in N(v)} \mathrm{MLP}_f\left(c^{(k)}(u)\right)\right)$$

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:



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

- We only need Φ to ensure the injectivity.

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

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

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

Differentiable color HASH function (injective)

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 similarity of different nodes in terms of their neighborhood structure.
  - Parameters of the update function can be **learned for the downstream tasks.**

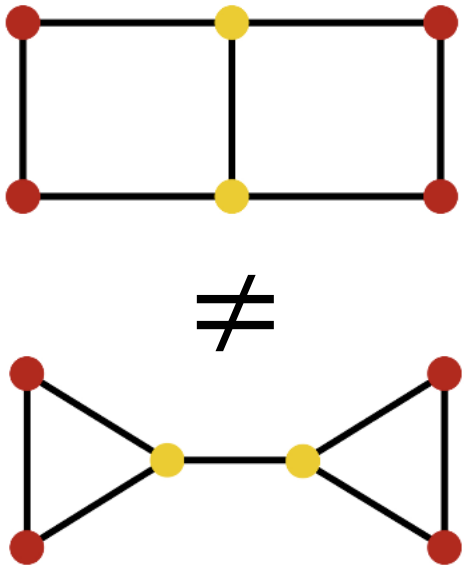
Expressive Power of GIN

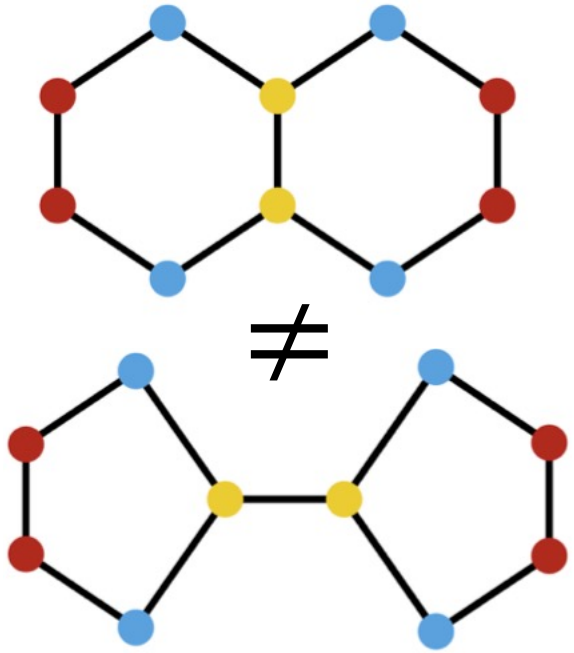
- **Because of the relation between GIN and the WL graph kernel, their expressive power is exactly the same.**
  - If two graphs can be distinguished by GIN, they can be also distinguished by the WL kernel, and vice versa.

- **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!

Failure cases of GIN

- **GIN is not perfect**.
  - fails in some cases where WL kernel fails as well (e.g., d-regular graphs, specific graphs contain ring / cycle structures…)
- **Examples** of non-isomorphic graphs that GIN fails to distinguish:





Today's Summary

- The expressive power of message-passing GNNs is **upper bounded by the WL isomorphism test.**
- GIN uses a neural network for neighbor aggregation function that can model **injective multi-set function**.
- GIN is closely related to the **WL graph kernel** and is **the most expressive GNN** model we introduced so far.
- The key is to use **element-wise sum pooling**, instead of mean-/max-pooling.