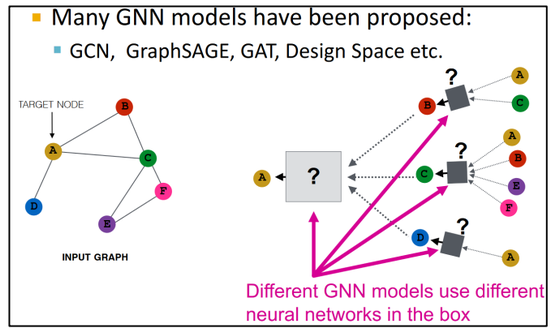
Contents

Expressive of GNN

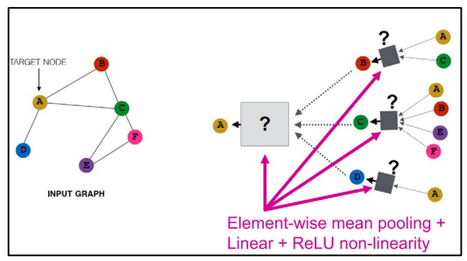
- Generate node embeddings based on **local network neighborhoods**
- Expressive power
  - ability to distinguish different graph structures of GNN

many GNN models

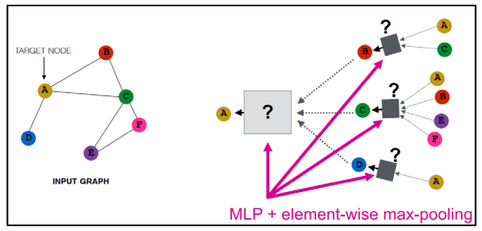


Example

GCN (mean-pool)

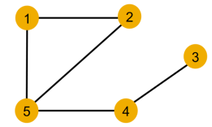


GraphSAGE (max-pool)



Node colors – assume same node features

- 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.

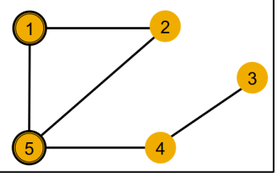


- **Key question: How well can a GNN distinguish different graph structures?**

Local neighbourhood structures

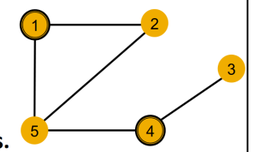
Node degree

- 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.



Same node degree, different neighbour node degree

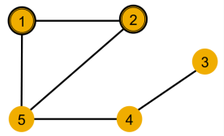
- **Example:** Nodes 1 and 4 both have the same node degree of 2. However, they still have **different** neighborhood structures because **their neighbors have different node degrees.**



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

Symmetric - same neighborhood structure

- **Example:** Nodes 1 and 2 have the **same** neighborhood structure because **they are symmetric within the graph.**

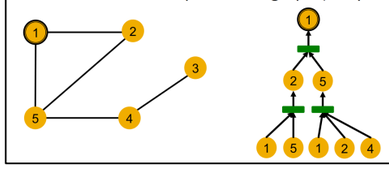


Node 1 has neighbors of degrees 2 and 3.
Node 2 has neighbors of degrees 2 and 3.
And even if we go a step deeper to 2nd hop neighbors, both nodes have the same degrees (Node 4 of degree 2)

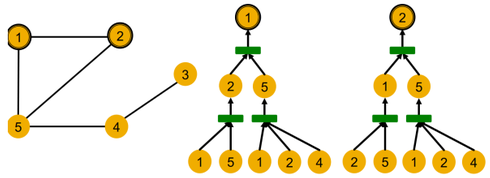
Computation graph

Ex - Node 1's computational graph (2-layer GNN)

- **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)



Node 1 & 2



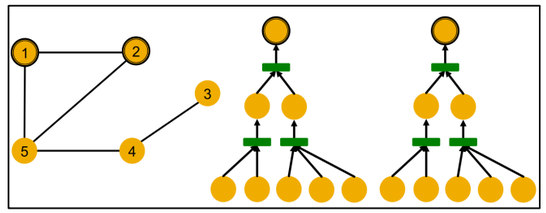
Same structure if ignore node features

- GNN won't be able to distinguish nodes 1 and 2
- Assumption
  - Node features are the same same color)

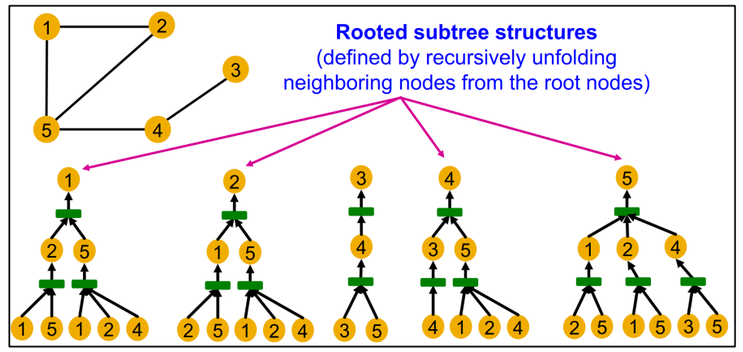
- If without node ID, then the computatino graphs for node 1 & 2 are the same (same color means same node features here)
- Meaning node 1 & 2 wil always be assined to the same class or label

- **A GNN will generate the same embedding 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.

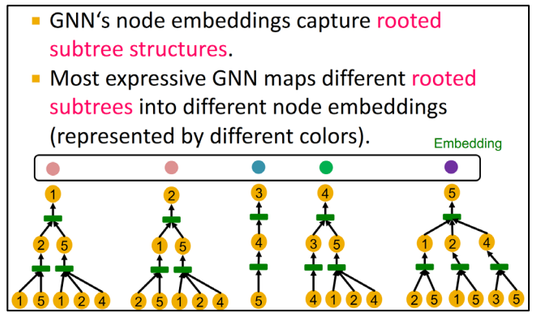


Computational graphs & rooted subtree structures



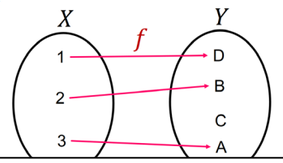
Expressiveness & rooted subtree structures

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

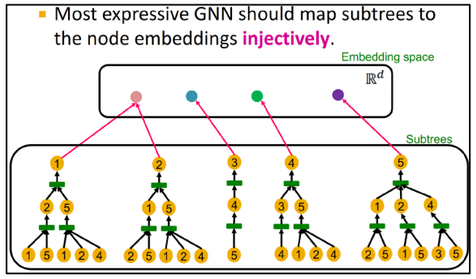


Injective function – each map to unique output

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

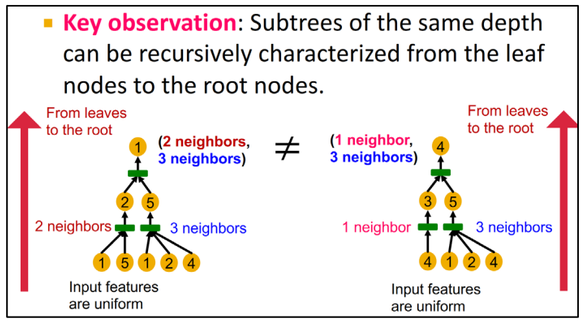


illustration



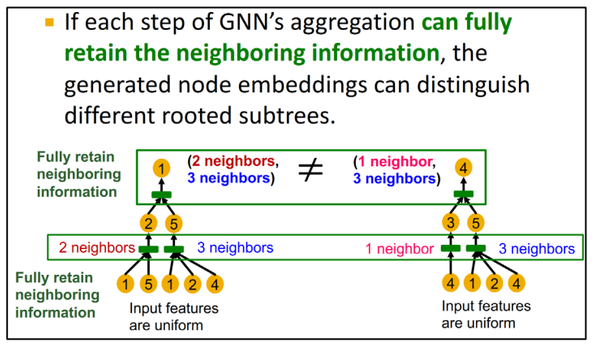
Observation

Message passing from leaf to root node

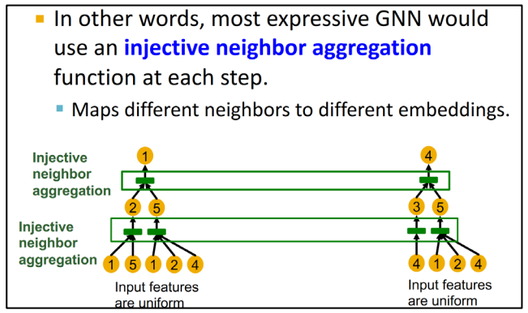


Retrain info from recursive different rooted subtrees

- Can differntiate subtree structure if different node has different number of neighbours
- Can we Aggregate info from the children and store it to pass it on to the parent so that the info is retrained
    - E.g. info about 2 & 3 #neighbours are passed to the root node 1
    - This helps store the structural info

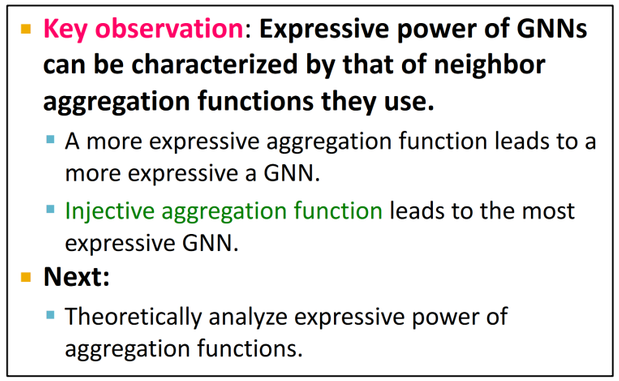


injective neighbor aggregation – if every steps

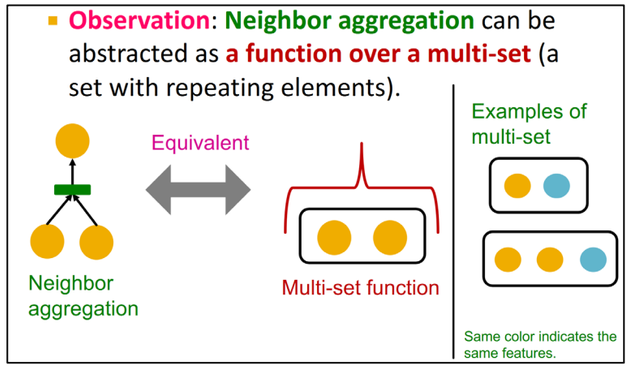


Lec 9.2

Expressiveness - neighbor aggregation functions

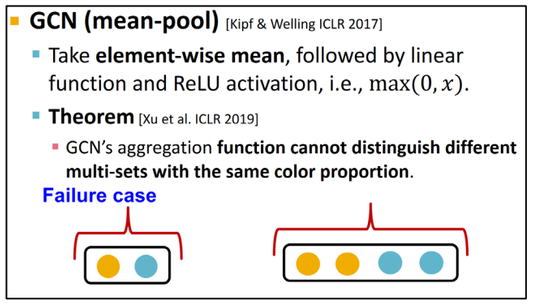


Neighbor aggregation - a function over a multi-set



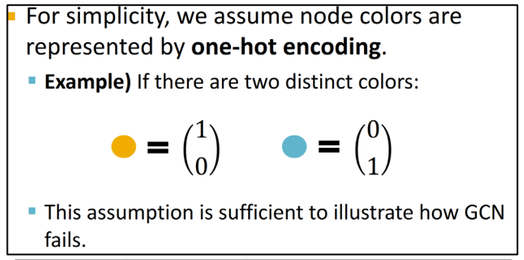
Case study – GCN & GraphSAGE

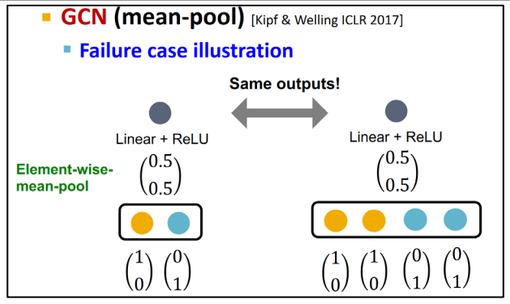
GCN



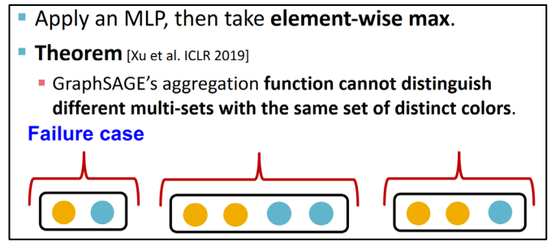
Failed case – multi-set with same color proportion

- Because the ratios are the same



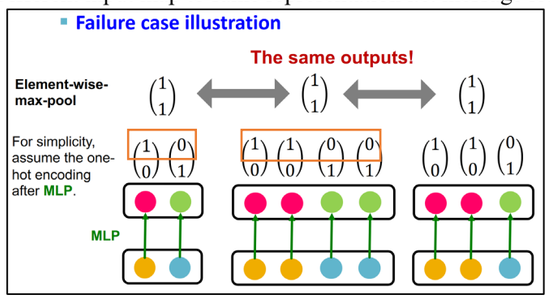


GraphSAGE (max-pool)



Failed illustration

- Different input map to same representation i.e. message loss



Designing most expressive GNNs – injective multiset function

- **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 – non-linear + sum over multi-set

Theorem [Xu et al. ICLR 2019]

Any injective multi-set function can be expressed as:

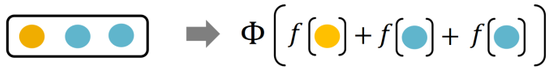
$$\Phi\left(\sum_{x\in S} f(x)\right)$$

Some non-linear function

Some non-linear function

Sum over multi-set

S : multi-set



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

Proof Intuition – f produces one hot, sum capture the global, non-linearity add expressive

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

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

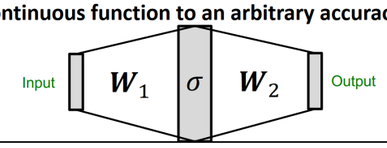
Example: $\Phi\left(f\left(\bigcirc\right)+f\left(\bigcirc\right)+f\left(\bigcirc\right)\right)$

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

Universarl approximation theorem – MLP w/ high dim & non-linearity

- **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**.



Injective multi-set function – dim = 100~500

- Linear tramsformation (weight matrix) + non-linearity (ReLu)
- Both learnt with MLP since universarl approximation i.e. can approximate any function
- Thus injective since can map different input to different output

- **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 – no fail case

- **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!**

GIN

- GIN by relating it to WL graph kernel (traditional way of obtaining graph-level features).

Relation to WL kernel

- Essentially – a hardcored GNN
  - Use hash function over MLP to map input to different ooutput
  - Aggregate the aggregation between nighbour and itself

Recall: 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

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

where HASH maps different inputs to different colors.

  - After K steps of color refinement, $c^{(K)}(v)$ summarizes the structure of K-hop neighborhood

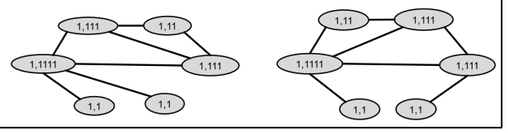
WL: color refinement

- Assume hash function is injective
- Process continues until a **stable coloring** is reached
- Two graphs are considered **isomorphic** if they have the **same set of colors**
- # of 1 is # neighbours

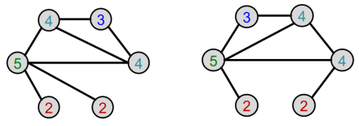
Example of color refinement given two graphs

- Assign initial colors

- Aggregate neighboring colors



- **Injectively** HASH the aggregated colors



HASH table: **Injective!**

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

GIN model

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

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

- 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

MLP aggregation, itself & neighbour

Theorem (Xu et al. ICLR 2019)

Any injective function over the tuple

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

Injective – one hot like, direct summation, MLP as one hot

- Direct summation with one hot vectors tells how many neighbours each node has with no info loss
- MLP serves like aggregate to a one hot so no info loss

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

Example: $\Phi\left(\bigcirc + \bigcirc + \bigcirc\right)$

$$\binom{1}{0} + \binom{0}{1} + \binom{0}{1} = \binom{1}{2}$$

- We only need Φ to ensure the injectivity.

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

Root node features — Neighboring node features — This MLP can provide "one-hot" input feature for the next layer.

Complete GIN model – GINConv over WL kernel

- **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

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

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 as differentiable 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.**

Lec summary

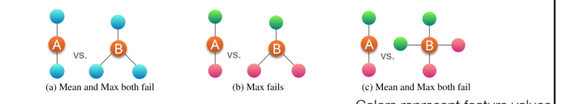
injective multi-set function, sum pooling, WL kernel

- 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!

Power of pooling

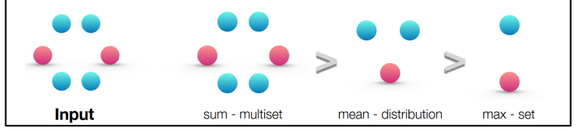
1. Same neighbour features
2. Disctinct colors are the same so whether the max is the same both cases
3. Same proportion

Failure cases for mean and max pooling:



Colors represent feature values

Ranking by discriminative power:



Expressive of GIN – similar to WL kernel

- **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!

Summary 2

- GNNs and connection to bijective functions on sets.
- Most powerful GNN is equivalent to WL graph isomorphism test.
- GIN is the most powerful GNN.
- Sum aggregator is more powerful than mean is more powerful than max.

My Summary

GNN capture **the local neighbour structure** through the **computation graph**, if the graphs are the same, then no differentiation from the embedding space

Why injective?

- Different embedding map to different output
- No info loss / collusions

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**.

Why use MLP for injective

- MLP can learn any function with approximate accuracy – universal approximation theorem

f & ϕ are some kinds of transformation convert to one hot so that can be better aggregated with sum with no info loss

Side

Element-wise – coordinate wise