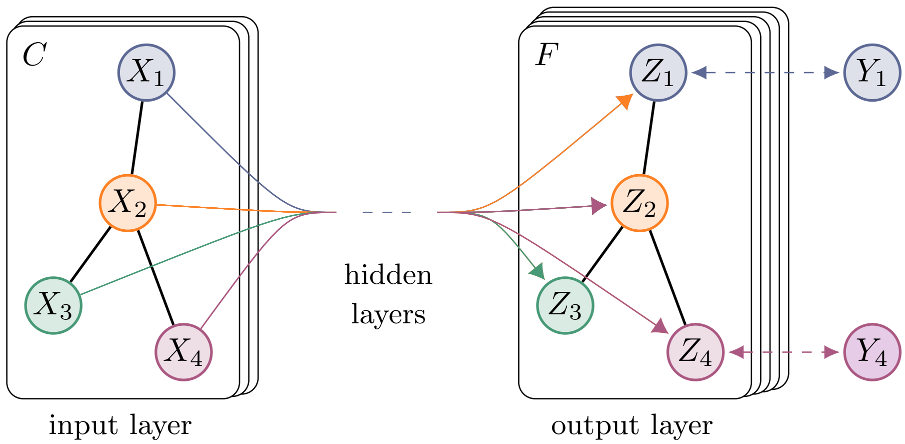
Neural Networks on Graph Data

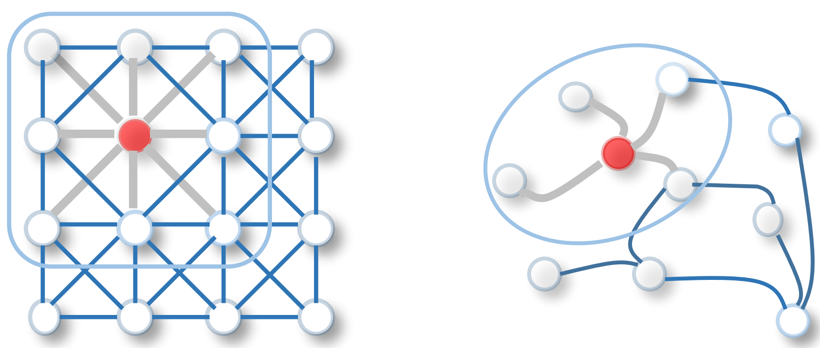


Features

- Allow to *learn* feature representations for nodes
- Allow to propagate information between neighbouring nodes
- Allow for efficient training (wrt to e.g. graph kernels)

Image from Kipf et al., 2017

Neural Networks on Graph Data



Basic step: graph "convolution"

- Aggregates information from neghbours to update information on node
- Inspired by convolution on pixels in CNN
- Differs from CNN convolution as neighbourhood has variable size

Image from Wu et al., 2019

Graph "convolution" operation

Generic form

- Aggregate information from neighbouring nodes:

$$h^{(k)}_{\mathcal{N}(v)} = \textsc{Aggregate}^{(k)}\left(\left\{h_u^{(k-1)} \ : \ u \in \mathcal{N}(v)\right\}\right)$$

- Combine node information with aggregated neighbour information:

$$h_v^{(k)} = \textsc{Combine}^{(k)}\left(h_v^{(k-1)}, h^{(k)}_{\mathcal{N}(v)}\right)$$

where

- k is the index of the layer (operations are layer-dependent)
- $h_v^{(k)}$ is the hidden representation of node v (initialized to the node features $h_v^{(0)} = x_v$)
- $\mathcal{N}(v)$ is the set neighbours of v

Example: GraphSAGE (Hamilton et al., 2017)

Graph "convolution" operation

- Mean aggregation

$$h^{(k)}_{\mathcal{N}(v)} = \textsc{mean}^{(k)}\left(\left\{h_u^{(k-1)} \ : \ u \in \mathcal{N}(v)\right\}\right)$$

- Max aggregation (on transformed representation)

$$h^{(k)}_{\mathcal{N}(v)} = \textsc{max}^{(k)}\left(\left\{\sigma\left(W^{(k)}_{pool} h_u^{(k-1)} + b\right) \ : \ u \in \mathcal{N}(v)\right\}\right)$$

- Combine operation as concatenation + linear mapping + non-linearity:

$$h_v^{(k)} = \sigma\left(W^{(k)}\left[h_v^{(k-1)}; h^{(k)}_{\mathcal{N}(v)}\right]\right)$$

Node embedding generation

Algorithm

1: $h_v^{(0)} = x_v \ \forall v \in \mathcal{V}$
2: **for** $k \in 1, \ldots, K$ **do**
3:     **for** $v \in \mathcal{V}$ **do**
4:         $h^{(k)}_{\mathcal{N}(v)} \leftarrow \textsc{Aggregate}^{(k)}\left(\left\{h_u^{(k-1)} \ : \ u \in \mathcal{N}(v)\right\}\right)$
5:         $h_v^{(k)} \leftarrow \textsc{Combine}^{(k)}\left(h_v^{(k-1)}, h^{(k)}_{\mathcal{N}(v)}\right)$
6:         $h_v^{(k)} \leftarrow h_v^{(k)} / ||h_v^{(k)}||$
7:     **end for**
8: **end for**
9: **return** $h_v^{(K)} \ \forall v \in \mathcal{V}$

Message Passing Neural Networks (MPNN)
Generic form

- Aggregate messages from neighbouring nodes:

$$m_v^{(k)} = \sum_{u \in \mathcal{N}(v)} M^{(k-1)}\left(h_v^{(k-1)}, h_u^{(k-1)}, e_{vu}\right)$$

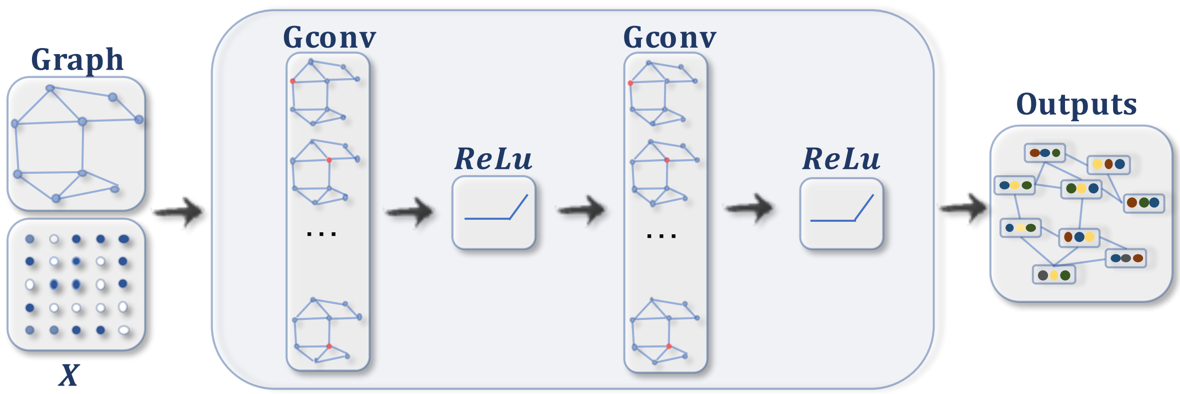
- Update node information:

$$h_v^{(k)} = U^{(k)}\left(h_v^{(k-1)}, m_v^{(k)}\right)$$

where

- e_{vu} are the features associated to edge (v, u)
- $M^{(k-1)}$ is a **message function** (e.g. an MLP) computing message from neighbour
- $U^{(k)}$ is a node **update function** (e.g. an MLP) combining messages and local information

Node Classification

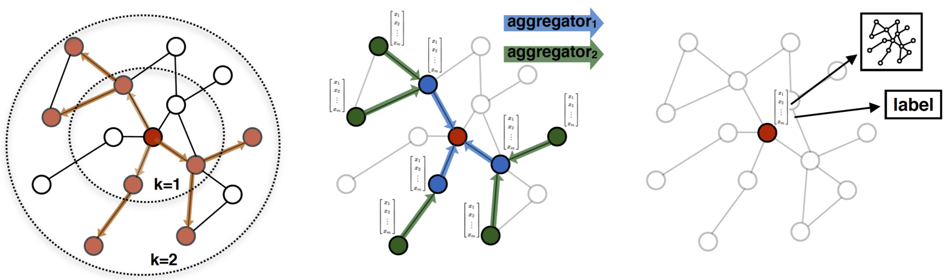


Procedure

- Compute node embeddings with layerwise architecture
- Add appropriate output layer on top of each node embedding (MLP + softmax, MLP + linear)

Image from Wu et al., 2019

Node classification: scalability



Sampling node neighbourhood

Replace $\mathcal{N}(v)$ with a layer-dependent sampling function $\mathcal{N}_k(v)$ that takes a random sample of a node's neighbourhood.

Image from Hamilton et al., 2017

GNN for graph classification

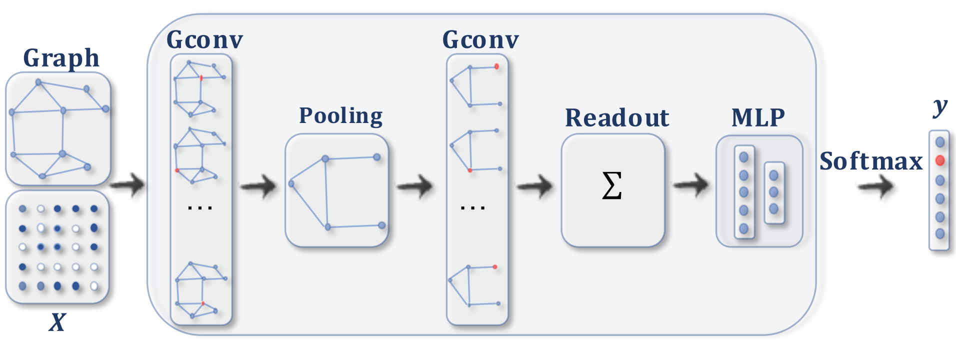
Basic approaches

- Apply final aggregation (READOUT) to combine all nodes in a single representation (mean, sum).
- Introduce a "virtual node" connected to all nodes in the graph

Problems

- No hierarchical structure is learned.
- Lack of "pooling" operation which is effective in CNNs to learn complex pattern.

Graph classification with Hierachical Pooling



Features

- Alternate convolutional and pooling layers as in CNN.
- Progressively reduce number of nodes.

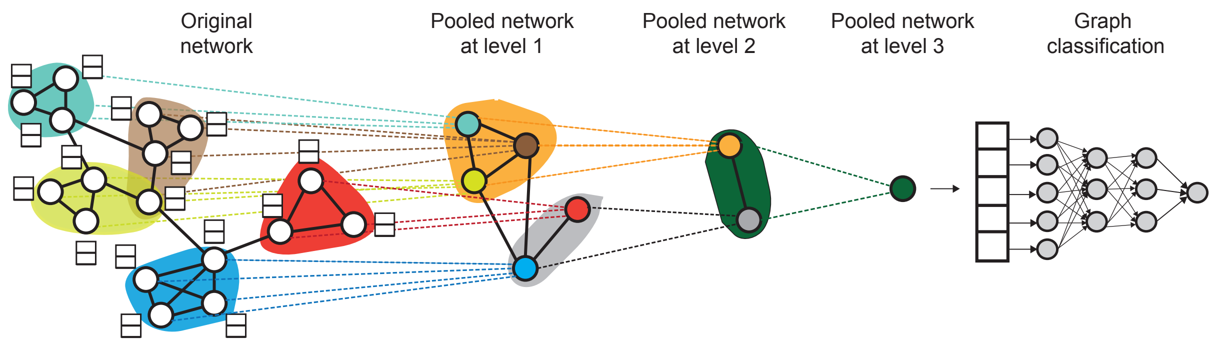
- Pool all nodes in last layer into a single representation.

Problem

How to decide which nodes to pool together

Image from Wu et al., 2019

Graph classification with Differentiable Pooling

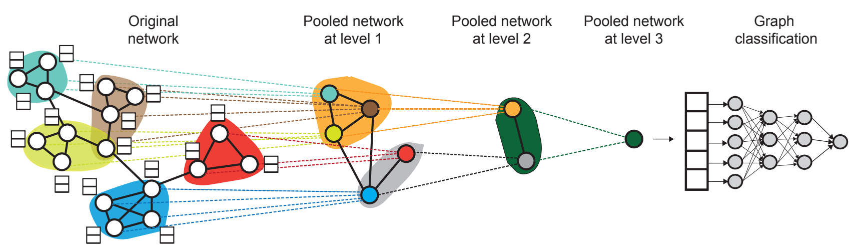


Idea

- Use standard GNN module to obtain embedding of nodes
- Perform graph pooling using a differentiable soft cluster assignment module
- Repeat the process for K layers
- Aggregate in single cluster in the last layer
- Use final representation to classify graph

Image from Ying et al., 2018

Graph classification with Differentiable Pooling



Components

- Layerwise soft cluster assignment matrix: $S^{(k)} \in \mathbb{R}^{n_k \times n_{k+1}}$
- Layerwise input embedding matrix: $Z^{(k)} \in \mathbb{R}^{n_k \times d}$
- Layerwise soft adjacency matrix: $A^{(k+1)}$
- Layerwise output embedding matrix: $X^{(k+1)} \in \mathbb{R}^{n_{k+1} \times d}$

Image from Ying et al., 2018

Graph classification with Differentiable Pooling

Compute $A^{(k+1)}, X^{(k+1)}$ given $S^{(k)}, Z^{(k)}$

- Computer $A^{(k+1)}$ based on connectivity strength between nodes in cluster

$$A^{(k+1)} = {S^{(k)}}^T A^{(k)} S^{(k)}$$

- Compute $X^{(k+1)}$ as weighted combination of cluster (soft) members

$$X^{(k+1)} = {S^{(k)}}^T Z^{(k)}$$

Graph classification with Differentiable Pooling

Compute $S^{(k)}, Z^{(k)}$ given $A^{(k)}, X^{(k)}$

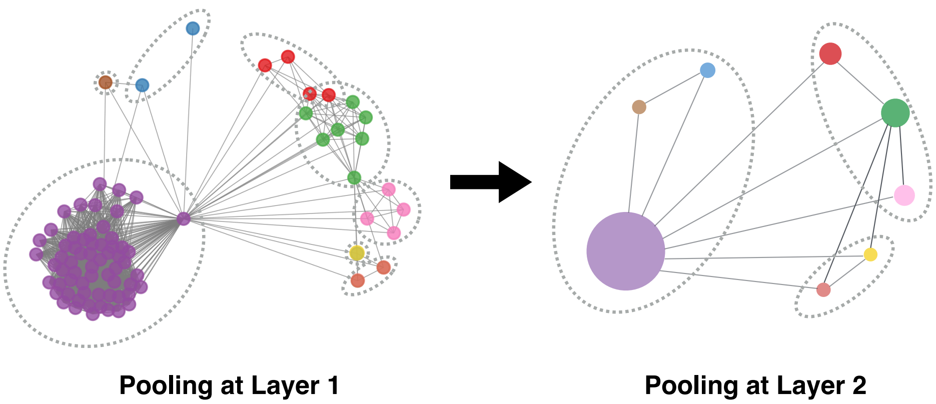
- Computer $Z^{(k)}$ using a standard GNN module

$$Z^{(k)} = \text{GNN}_k^{embed}(A^{(k)}, X^{(k)})$$

- Computer $S^{(k)}$ using a second standard GNN module followed by a per-row softmax

$$S^{(k)} = \textsc{softmax}\left(\text{GNN}_k^{pool}(A^{(k)}, X^{(k)})\right)$$

Graph classification with Differentiable Pooling



Note

The maximal number of clusters in the following layer (n_{k+1}) is a hyper-parameter of the model (typically 10-25% of n_k).

Image from Ying et al., 2018

Graph classification with Differentiable Pooling

Side objectives

Training using only graph classification loss can be difficult (very indirect signal). Two side objectives are introduced at each layer k:

link prediction Encourage nearby nodes to be pooled together:

$$L_{LP} = ||A^{(k)} - S^{(k)}S^{(k)^T}||_F$$

where $||M||_F = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{m}|M_{i,j}|^2}$

cluster entropy Encourage hard assignment of nodes to clusters:

$$L_E = \frac{1}{n_k}\sum_{i=1}^{n_k} H(S_i^{(k)})$$

where $H(S_i^{(k)})$ is the entropy of the i^{th} row of $S^{(k)}$.

Attention Mechanisms for GNN

What is Attention

- Attention is a mechanism that allows a network to focus on certain parts of the input when processing it
- In multi-layered networks attention mechanisms can be applied at all layers
- It is useful to deal with variable-sized inputs (e.g. sequences)

Attention Mechanisms for GNN

Why Attention in GNN

- GNN compute node representations from representations of neighbours
- Nodes can have largely different neighbourhood sizes
- Not all neighbours have relevant information for a certain node
- Attention mechanism allow to adaptively *weight* the contribution of each neighbour when updating a node

Graph Attention Networks (GAT)

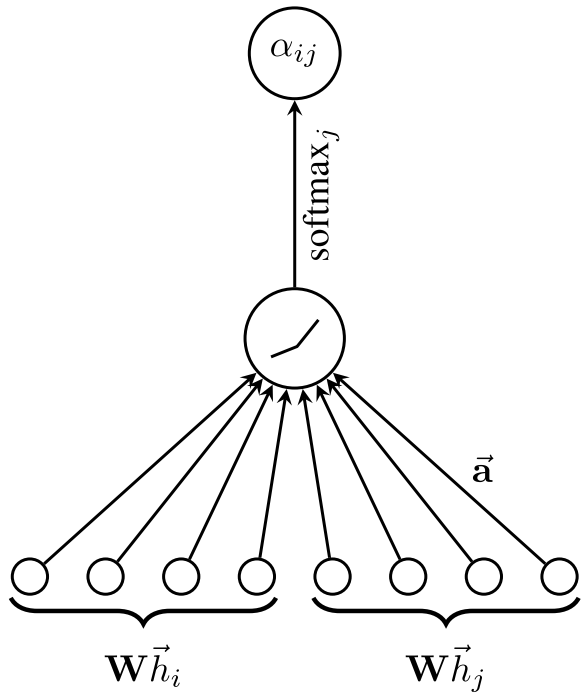
Attention coefficients

$$\alpha_{ij} = \frac{f(Wh_i, Wh_j)}{\sum_{j' \in \mathcal{N}(i)} f(Wh_i, Wh_{j'})}$$

- Models importance of node j for i as a function of their representations
- Node representations are first transformed using W
- An attentional mechanism f, shared for all nodes computes attention of i for j
- Attention coefficient is normalized over neighbours of i (including i itself)

Graph Attention Networks (GAT)

Image from Veličković, et al., 2018



Attention mechanism
$f(Wh_i, Wh_j) = \text{LeakyReLU}\left(a^T [Wh_i; Wh_j]\right)$

Graph Attention Networks (GAT)

Node update

$$h_i^{(k)} = \sigma\left(\sum_{j \in \mathcal{N}(i)} \alpha_{ij} W h_j^{(k-1)}\right)$$

- Node is updated as the sum of neighbour (updated) representations, each weighted by its attention coefficient
- A non-linearity σ is (possibly) applied to this updated representation

Graph Attention Networks (GAT)

Multi-head attention

$$h_i^{(k)} = \text{CONCAT}\left[\sigma\left(\sum_{j \in \mathcal{N}(i)} \alpha_{ij}^{\ell} W^{\ell} h_j^{(k-1)}\right) \middle| \ell = 1, \ldots, L\right]$$

- Multi-head attention works by having multiple (L) simultaneous attention mechanisms
- Can be beneficial to stabilize learning (see Transformers)
- Updated node representation is concatenation of representations from different heads.
- CONCAT is replaced by MEAN in output layer

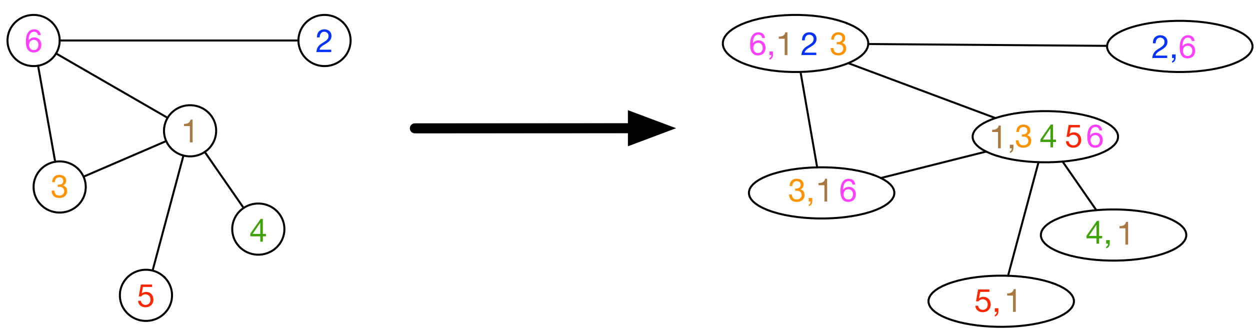
Representational power of GNN

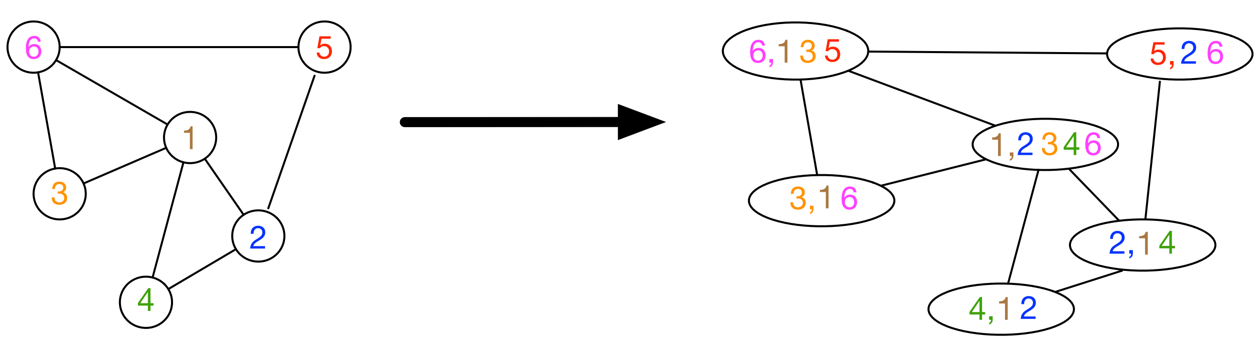
Weistfeiler-Lehman (WL) isomorphism test

Given $G = (\mathcal{V}, \mathcal{E})$ and $G' = (\mathcal{V}', \mathcal{E}')$, with $n = |\mathcal{V}| = |\mathcal{V}'|$. Let $L(G) = \{l(v)|v \in \mathcal{V}\}$ be the set of labels in G, and let $L(G) == L(G')$. Let $label(s)$ be a function assigning a unique label to a string.

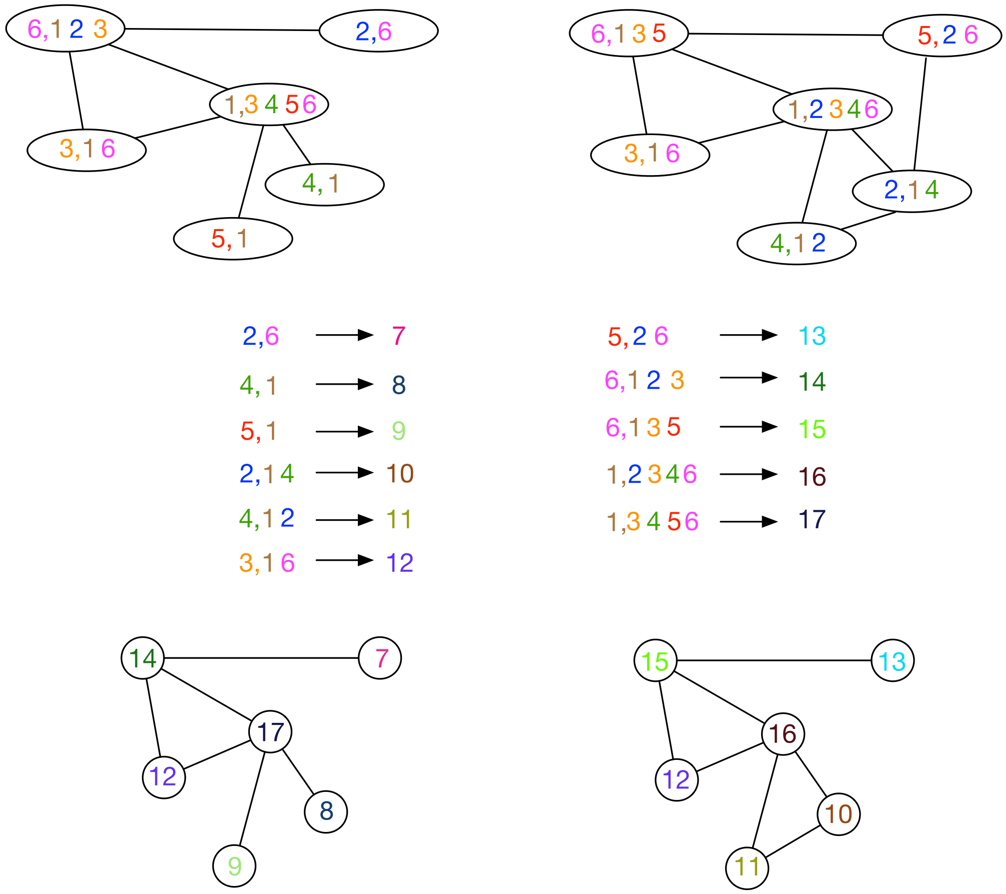
- Set $l_0(v) = l(v)$ for all v.
- For $i \in [1, n-1]$
  1. For each node v in G and G'
  2. Let $M_i(v) = \{l_{i-1}(u)|u \in neigh(v)\}$
  3. Concatenate the sorted labels of $M_i(v)$ into $s_i(v)$
  4. Let $l_i(v) = label(l_{i-1}(v) \circ s_i(v))$ ($\circ$ is concatenation)
  5. If $L_i(G) \mathcal{N} L_i(G')$
  6. Return **Fail**
- Return **Pass**

WL isomorphism test: string determination





WL isomorphism test: relabeling



Representational power of GNN

Theorem (Xu et al., 2019)

Let $\mathcal{F} : \mathcal{G} \to \mathbb{R}^d$ be a GNN. With enough GNN layers, $\mathcal{F}$ maps any graphs G_1 and G_2 judged non-isomorphic by the Weisfeiler-Lehman test to different embeddings if:

- $\mathcal{F}$ aggregates and updates node features iteratively with

  $$h_v^{(k)} = \phi\left(h_v^{(k-1)}, f\left(\left\{h_u^{(k-1)} : u \in \mathcal{N}(v)\right\}\right)\right)$$

  where f and ϕ are injective functions

- $\mathcal{F}$ computes the graph-level readout using an injective function over node features $\left\{h_v^{(k)}\right\}$

Note

No (first-order) GNN can have a higher representational power than the Weisfeiler-Lehman test of isomorphism.

Representational power of GNN

Corollary (simplified)
Any function $g(c, X)$ with $c \in \mathcal{X}$ and $X \subset \mathcal{X}$ can be decomposed as:

$$g(c, X) = \phi\left((1+\epsilon)f(c) + \sum_{x \in X} f(x)\right)$$

for some functions f and ϕ and infinitely many choices of ϵ

Problem

- Assumes countable $\mathcal{X}$ (no real values).
- Leverages universal approximation theorem of MLPs, learnability can be hard in practice.

Graph Isomorphism Networks (GIN)

Definition

- Update node representation by:

$$h_v^{(k)} = \text{MLP}^{(k)}\left((1+\epsilon^{(k)})h_v^{(k-1)} + \sum_{u \in \mathcal{N}(v)} h_u^{(k-1)}\right)$$

- Compute graph readout as:

$$h_G = \text{CONCAT}\left(\sum_{v \in G} h_v^{(k)} \,\middle|\, k = 0, \ldots, K\right)$$

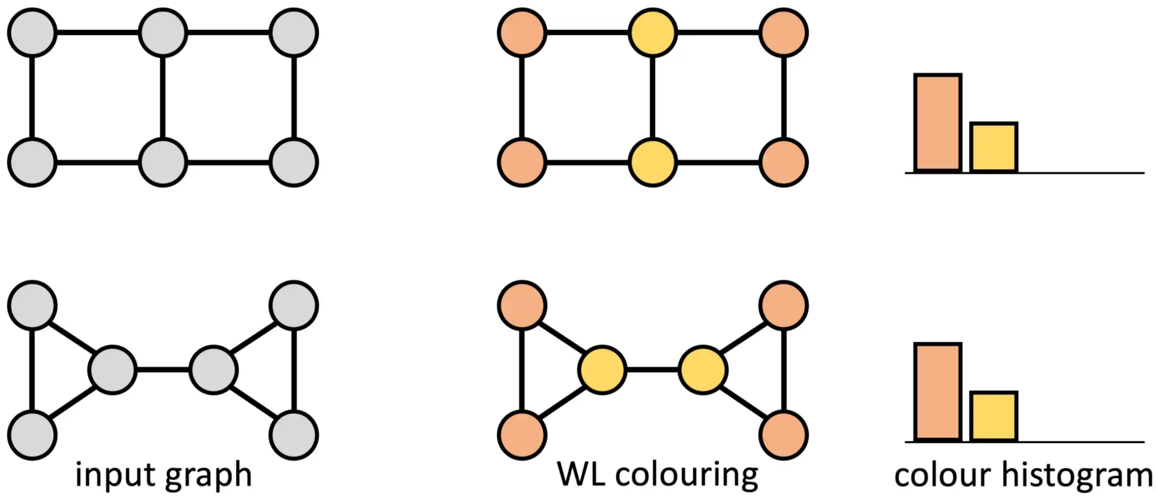
Note
Definition guarantees maximal representational power achievable for a GNN (other choices are possible)

Graph Isomorphism Networks (GIN)

Notes

- The $\text{MLP}^{(k)}$ jointly models $f^{(k+1)} \circ \phi^{(k)}$ (universal approximator)
- $\epsilon^{(k)}$ can be replaced by a fixed scalar
- CONCAT is used to collect all structural information. It could be replaced by the latest representation (layer K).

Representational power of GNN

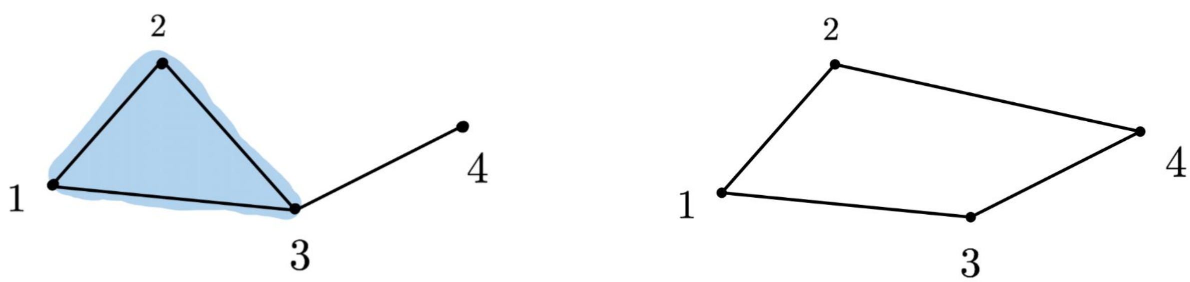


Limitations of the WL isomorphism test

- The WL isomorphism test is limited in the graph substructures it can count
- The WL isomorphism test fails to recognize the two input graphs as non-isomorphic

Image from Bronstein, 2021

Higher-order GNN



$K = \{1, 2, 3, 4, 12, 13, 23, 34, 123\}$ $K = \{1, 2, 3, 4, 12, 13, 24, 34\}$

Simplician complex

- A *simplex* is the generalization of a triangle to arbitrary dimensions (0=point, 1=line, 2=triangle, 3=tetrahedron, ..)
- A *simplicial complex* K is a set of simplices such that:
  - Every face of a simplex from K is also in K
  - The non-empty intersection of any two simplices $\sigma_1, \sigma_2 \in K$ is a face of both σ_1 and σ_2.

Images (from here onwards) from Bodnar et al., 2021

Higher-order GNN

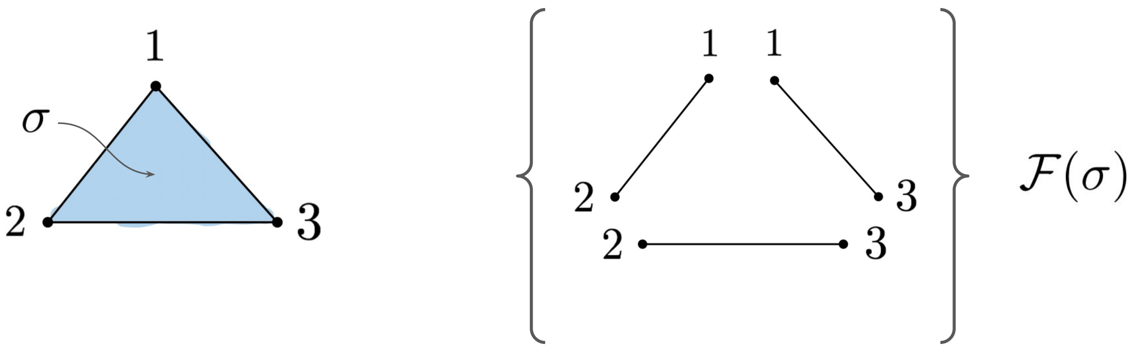
Simplician Weisfeiler-Lehman (SWL) Test

Let K be a simplicial complex. SWL proceeds as follows:

1. Assign each simplex $s \in K$ an initial colour.
2. Compute the new colour of each simplex s by hashing the concatenation of its color and the colours of its neighbouring simplices.
3. Repeat until a stable coloring is obtained

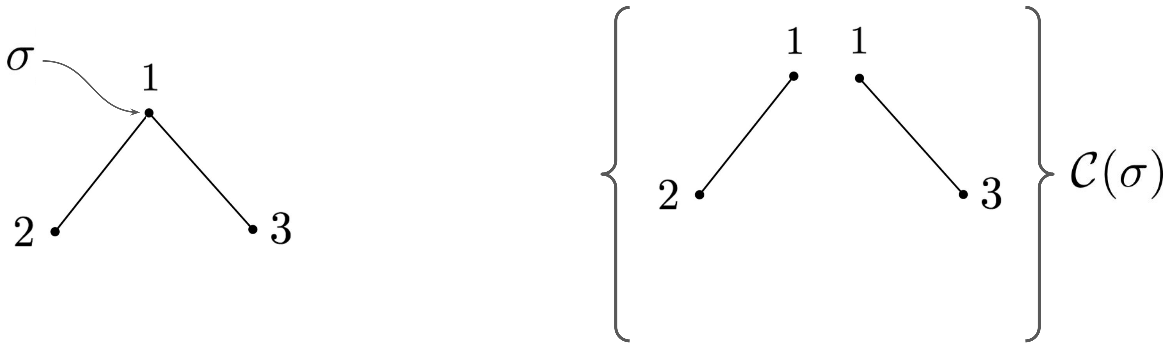
Two simplicial complexes are considered non-isomorphic if the colour histograms at any level of the complex are different.

Types of adjacencies: face adjacencies



$$c^t_{\mathcal{F}}(\sigma) = \underbrace{\{\!\!\{ c^t_\omega \mid \omega \in \overbrace{\mathcal{F}(\sigma)}^{\text{set of faces}} \}\!\!\}}_{\text{multiset of face colours}}$$

Types of adjacencies: coface adjacencies

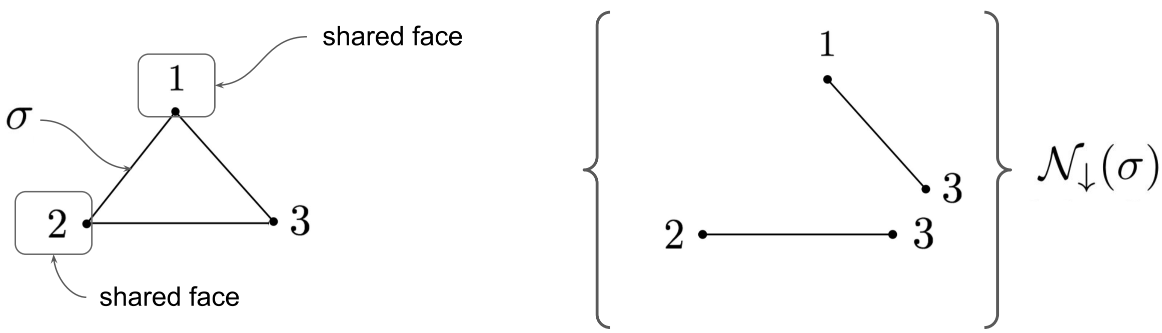


set of cofaces

$$c^t_{\mathcal{C}}(\sigma) = \underbrace{\{\!\{c^t_\omega | \omega \in \overbrace{\mathcal{C}(\sigma)}^{\text{set of cofaces}}\}\!\}}_{\text{multiset of coface colours}}$$

multiset of coface colours

Types of adjacencies: lower adjacencies



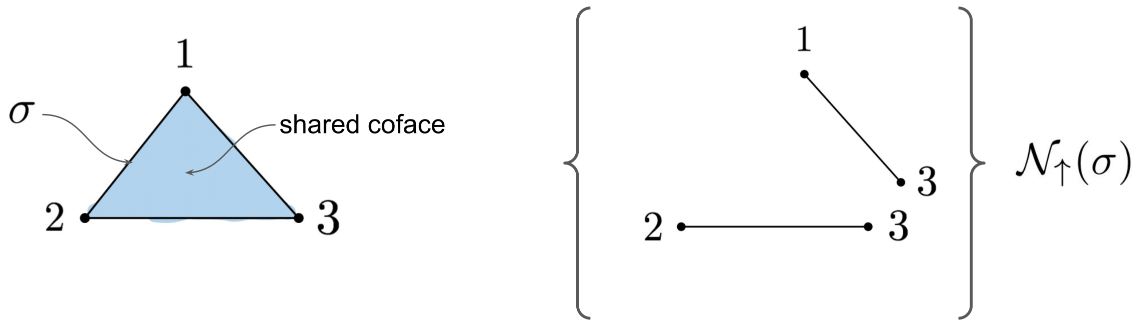
set of lower-neighbours

$$c^t_{\downarrow}(\sigma) = \underbrace{\{\!\{(c^t_\omega, c^t_{\sigma\cap\omega}) | \omega \in \overbrace{\mathcal{N}_{\downarrow}(\sigma)}^{\text{set of lower-neighbours}}\}\!\}}_{\text{multiset of lower-neighbours colour-tuples}}$$

multiset of lower-neighbours colour-tuples

Two *d*-simplices are lower adjacent if they share a common face of dimension *d-1*

Types of adjacencies: upper adjacencies

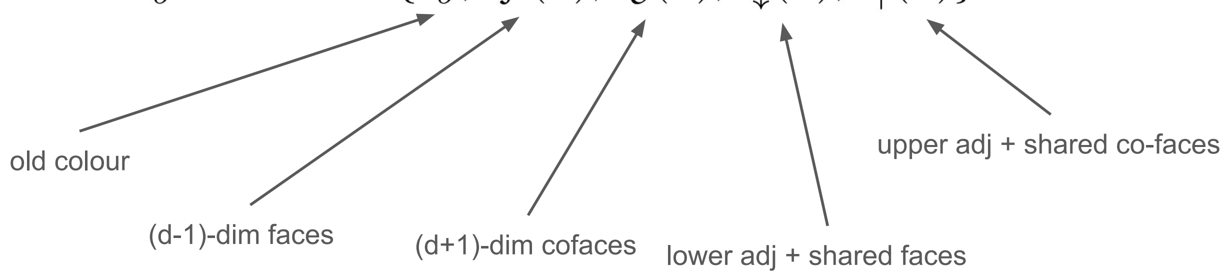


$$c^t_\uparrow(\sigma) = \underbrace{\{\!\{(c^t_\omega, c^t_{\sigma\cup\omega}) | \omega \in \overbrace{\mathcal{N}_\uparrow(\sigma)}^{\text{set of upper-neighbours}}\}\!\}}_{\text{multiset of upper-neighbours colour-tuples}}$$

Two *d*-simplices are upper adjacent if they share a common coface of dimension *d+1*

SWL coloring

$$c^{t+1}_\sigma = \mathrm{HASH}\{c^t_\sigma, c^t_{\mathcal{F}}(\sigma), c^t_{\mathcal{C}}(\sigma), c^t_\downarrow(\sigma), c^t_\uparrow(\sigma)\}$$



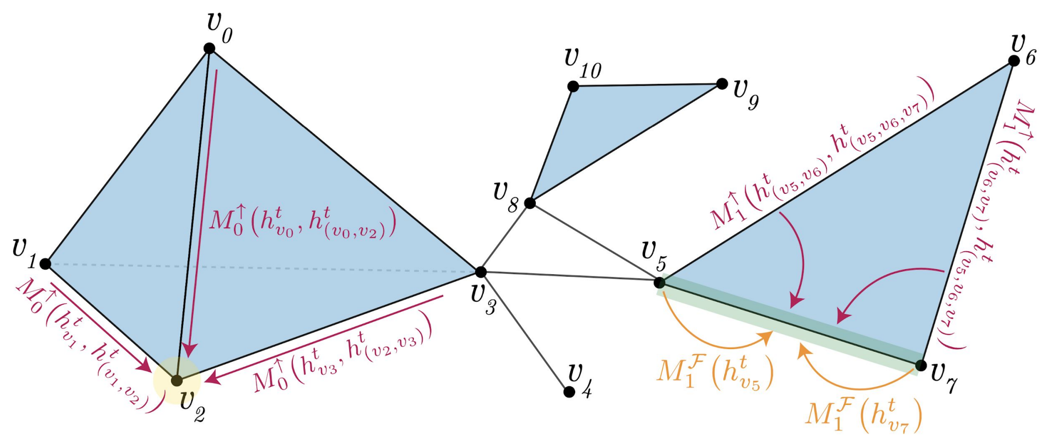
Message Passing Simplician Networks

$$
\left.\begin{aligned}
m_{\mathcal{F}}^{t+1}(v) &= \mathrm{AGG}_{w\in\mathcal{F}(v)}\Big(M_{\mathcal{F}}\big(h_v^t, h_w^t\big)\Big)\\
m_{\mathcal{C}}^{t+1}(v) &= \mathrm{AGG}_{w\in\mathcal{C}(v)}\Big(M_{\mathcal{C}}\big(h_v^t, h_w^t\big)\Big)\\
m_{\downarrow}^{t+1}(v) &= \mathrm{AGG}_{w\in\mathcal{N}_\downarrow(v)}\Big(M_{\downarrow}\big(h_v^t, h_w^t, h_{v\cap w}^t\big)\Big)\\
m_{\uparrow}^{t+1}(v) &= \mathrm{AGG}_{w\in\mathcal{N}_\uparrow(v)}\Big(M_{\uparrow}\big(h_v^t, h_w^t, h_{v\cup w}^t\big)\Big)
\end{aligned}\right\}\ \textbf{Message \& Aggregate}
$$

$$
h_v^{t+1} = U\Big(h_v^t, m_{\mathcal{F}}^t(v), m_{\mathcal{C}}^t(v), m_{\downarrow}^{t+1}(v), m_{\uparrow}^{t+1}(v)\Big) \quad \Big\}\ \textbf{Update}
$$

$$
h_G = \mathrm{READOUT}(\{\!\{h_v^L\}\!\}_{v\in\mathcal{K}_0}, \ldots, \{\!\{h_v^L\}\!\}_{v\in\mathcal{K}_p}) \quad \Big\}\ \textbf{Readout}
$$

Message Passing Simplician Networks



Message passing examples

- Messages from upper adjacencies for vertex v_2
- Messages from upper and face adjacencies for edge (v_5, v_7)

References

Bibliography

- Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, Philip S. Yu, *A Comprehensive Survey on Graph Neural Networks*, ArXiv, 2019.

- William L. Hamilton, Rex Ying, Jure Leskovec, *Inductive Representation Learning on Large Graphs*. In NIPS 2017.
- J. Gilmer, S. Schoenholz, P. Riley, O. Vinyals, and G. Dahl, *Neural message passing for Quantum chemistry*. In ICML 2017.
- R. Ying, J. You, C. Morris, X. Ren, W. L. Hamilton, and J. Leskovec, *Hierarchical graph representation learning with differentiable pooling*. In NIPS, 2018.
- Keyulu Xu, Weihua Hu, Jure Leskovec, Stefanie Jegelka, *How Powerful are Graph Neural Networks?*. In ICLR, 2019.
- P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò and Y. Bengio, *Graph Attention Networks*. In ICLR, 2018.
- C. Bodnar, F. Frasca, N. Otter, Y. Wang, P. Liò, G. Montufar, M. Bronstein, *Weisfeiler and Lehman Go Topological: Message Passing Simplical Networks*. In NeurIPS, 2021.

References

Software Libraries

- PyTorch Geometric (PyG) [`https://github.com/pyg-team/pytorch_geometric`]
- Deep Graph Library (dgl) [`www.dgl.ai`]