Aggregation Functions in GNNs

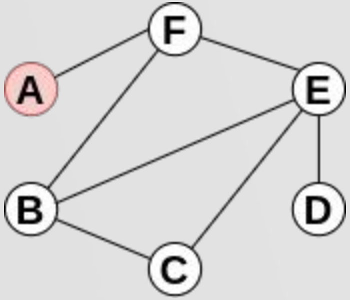
Giovanni Pellegrini[1,2,3]

SML[1] Lab, University of Trento, Italy
TIM[2]
EIT DIGITAL[3]

COMPUTATION GRAPH

The neighbour of a node defines its computation graph

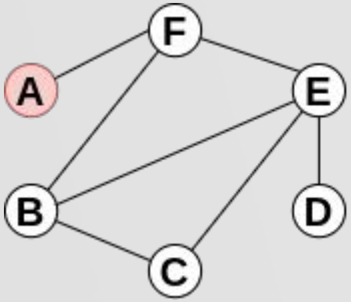
INPUT GRAPH



COMPUTATION GRAPH

The neighbour of a node defines its computation graph

INPUT GRAPH



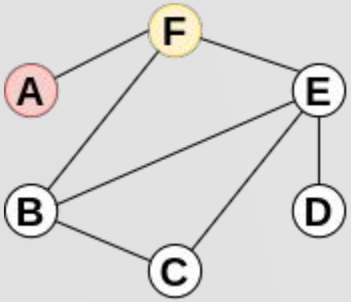
COMPUTATION GRAPH

01 Recap

COMPUTATION GRAPH

The neighbour of a node defines its computation graph

INPUT GRAPH

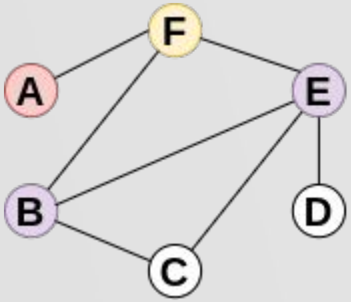


COMPUTATION GRAPH

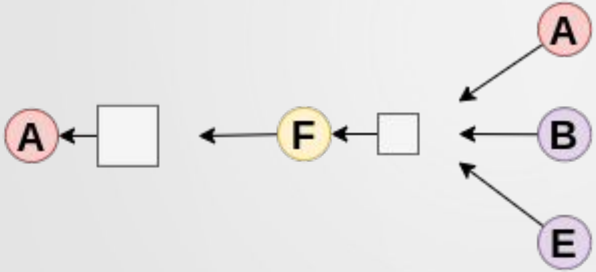
COMPUTATION GRAPH

The neighbour of a node defines its computation graph

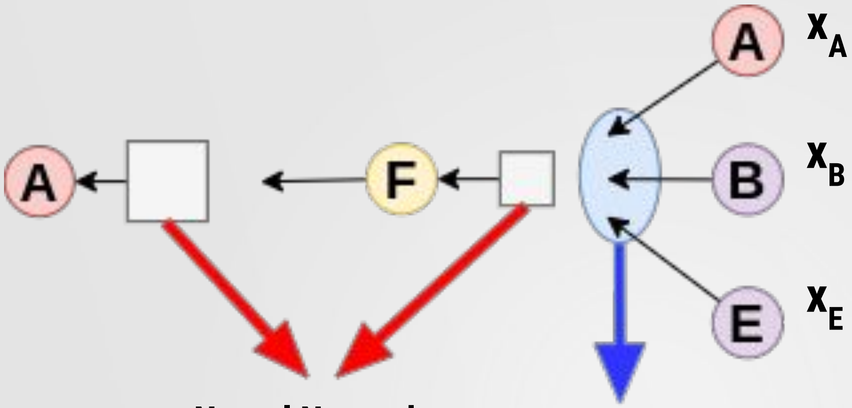
INPUT GRAPH



COMPUTATION GRAPH



01 Recap



Neural Networks

<u>Permutation invariant</u>
Aggregation

Sum
Average
Max

01 Recap

GCN	mean
GraphSage	max, mean, LSTM
GAT	sum

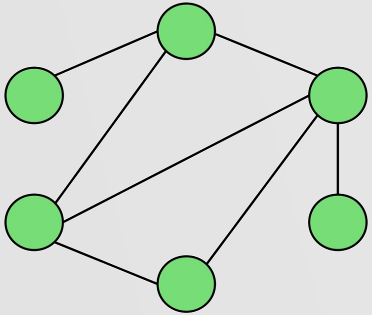
TABLE OF CONTENTS

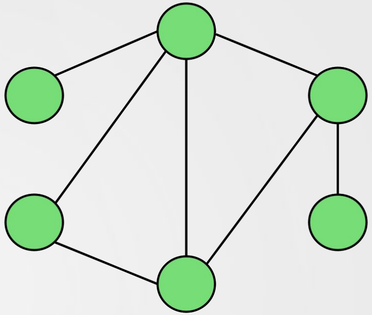
02 WL Isomorphism Test

Solution: Weisfeiler-Lehman isomorphism test[1]

[1] Weisfeiler and Lehman. *A reduction of a graph to a canonical form and an algebra arising during this reduction.* Nauchno-Technicheskaya Informatsia, 1968.

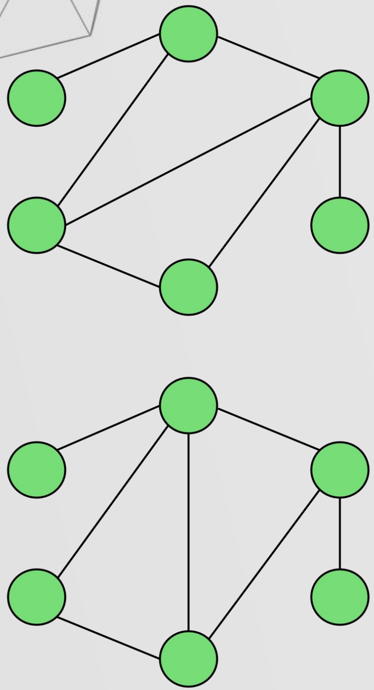
02 WL Isomorphism Test





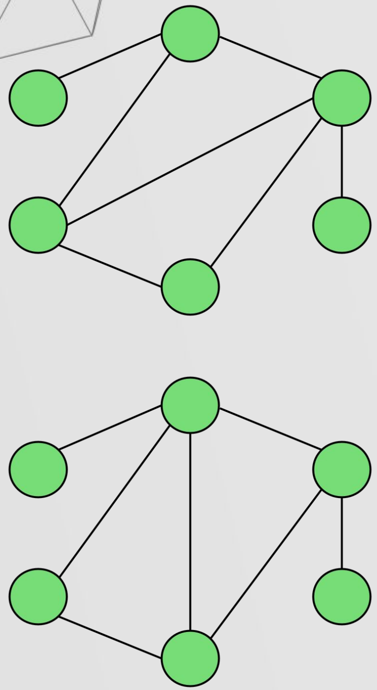
Step 0

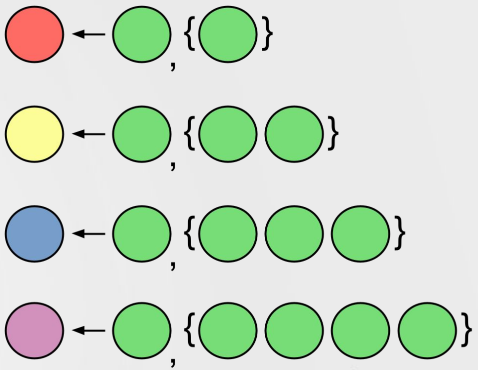
02 WL Isomorphism Test



Step 1

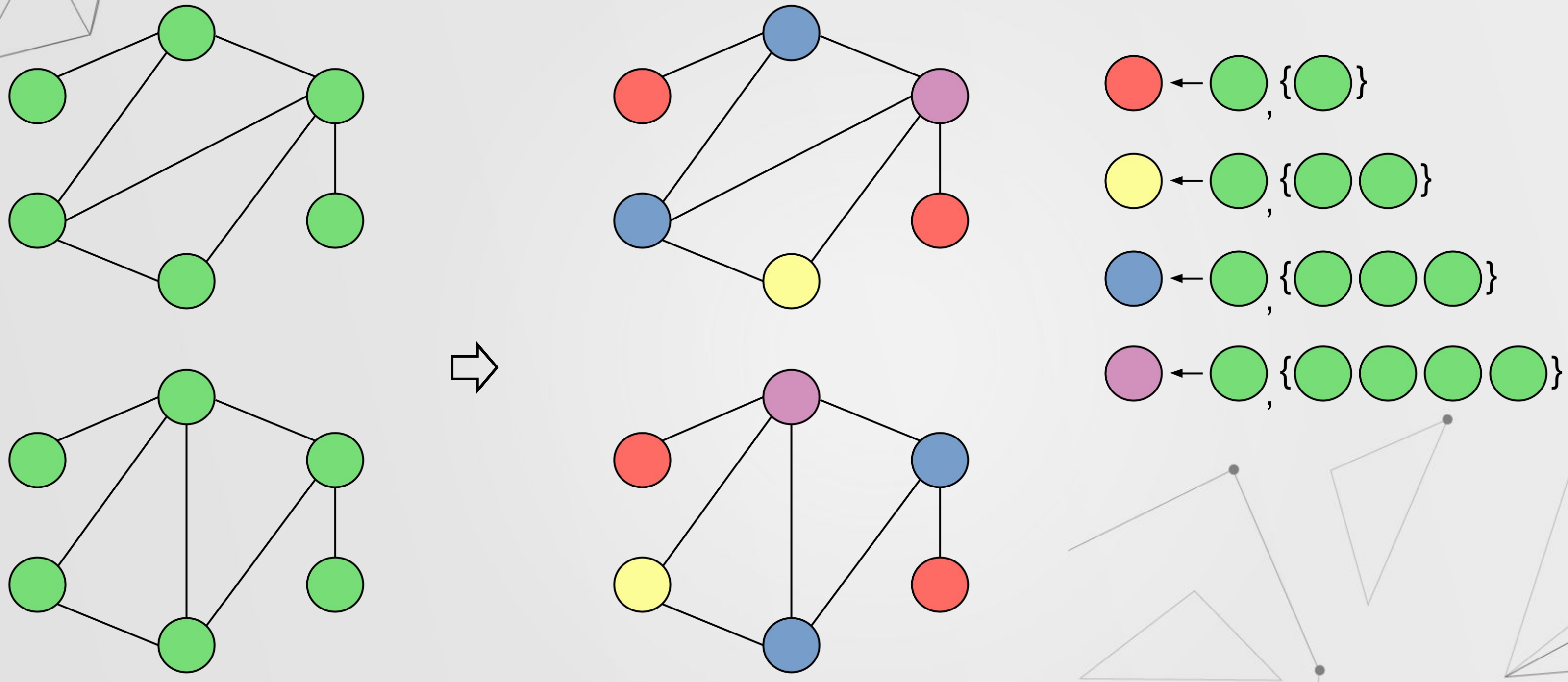
02 WL Isomorphism Test





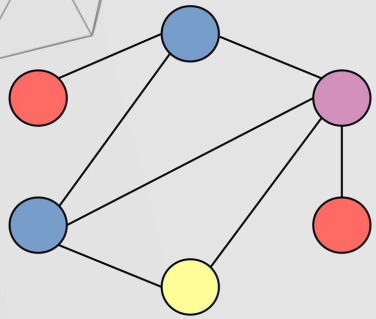
Step 1

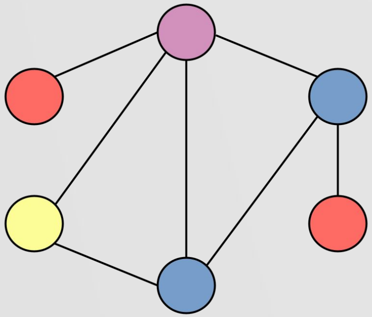
02 WL Isomorphism Test



Step 1

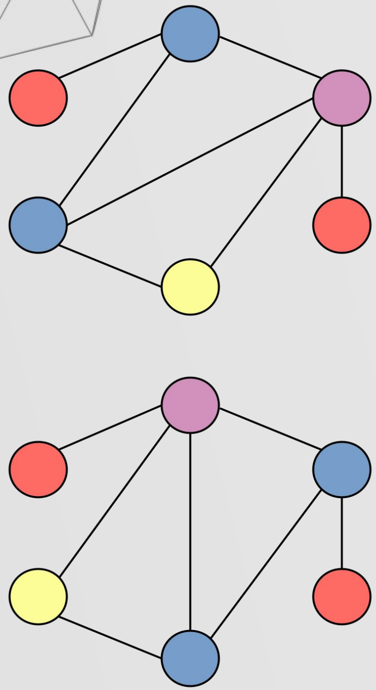
02 WL Isomorphism Test

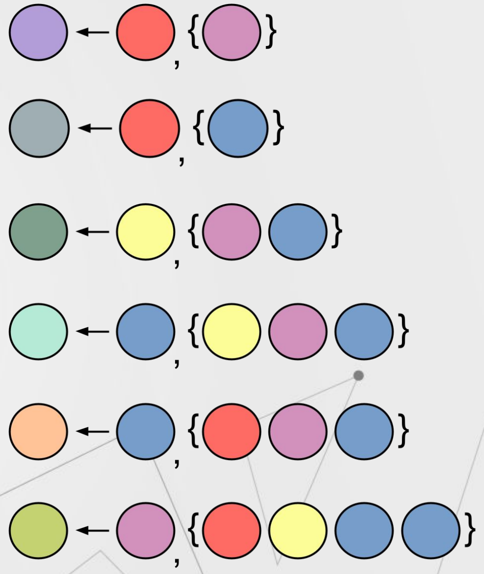




Step 2

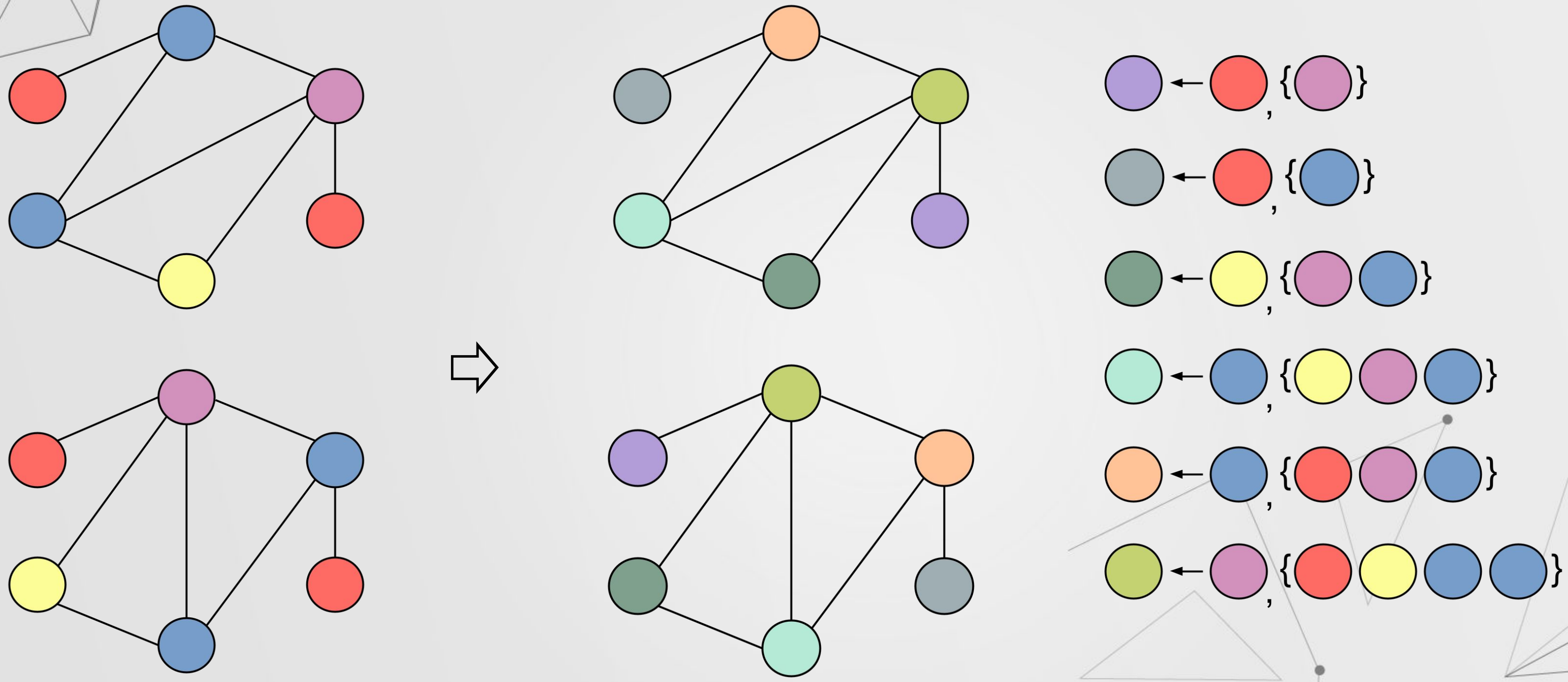
02 WL Isomorphism Test





Step 2

02 WL Isomorphism Test



Step 2

02 WL Isomorphism Test

$$c_i^{(k)} = h\left(c_i^{(k-1)}, \{c_j^{(k-1)} : j \in \mathcal{N}(i)\}\right)$$

02 WL Isomorphism Test

$$c_i^{(k)} = h\left(\underbrace{c_i^{(k-1)}}_{\text{Observed node}}, \underbrace{\{c_j^{(k-1)} : j \in \mathcal{N}(i)\}}_{\text{Neighbours' color}}\right)$$

02 WL Isomorphism Test

$$c_i^{(k)} = \underbrace{\boxed{h}}_{\text{Injective function}}\Big(\underbrace{\boxed{c_i^{(k-1)}}}_{\text{Observed node}}, \underbrace{\boxed{\{c_j^{(k-1)} : j \in \mathcal{N}(i)\}}}_{\text{Neighbours' color}}\Big)$$

02 WL Isomorphism Test

$$c_i^{(k)} = \underbrace{\boxed{h}}_{\text{Injective function}}\Big(\underbrace{\boxed{c_i^{(k-1)}}}_{\text{Observed node}}, \underbrace{\boxed{\{c_j^{(k-1)} : j \in \mathcal{N}(i)\}}}_{\text{Neighbours' color}}\Big)$$

- Efficient heuristic
- Isomorphic graphs -> same labels
- Nodes are uniquely coloured
- Distinguish most graphs

But… limited use in practice

03 Graph Isomorphism Network (GIN)

Can we construct a GNNs as powerful as the WL isomorphism test?

03 Graph Isomorphism Network (GIN)

Can we construct a GNNs as powerful as the WL isomorphism test?

GIN - Graph Isomorphism Network[2]

[2] Xu et al., *How powerful are graph neural networks?*, International Conference on Learning Representations, 2019

03 Graph Isomorphism Network (GIN)

G, G' two non-isomorphic graphs

$\mathcal{A} : G \to \mathbb{R}^d$ a GNN

03 Graph Isomorphism Network (GIN)

G, G' two non-isomorphic graphs

$\mathcal{A} : G \to \mathbb{R}^d$ a GNN

Construct $\mathcal{A}$ s.t. $\{h_i : i \in V(G)\}$ and $\{h_j : j \in V(G')\}$ differ

$\longrightarrow$ WL test decides they are non-isomorphic

03 Graph Isomorphism Network (GIN)

G, G' two non-isomorphic graphs

$\mathcal{A} : G \to \mathbb{R}^d$ a GNN

Construct $\mathcal{A}$ s.t. $\{h_i : i \in V(G)\}$ and $\{h_j : j \in V(G')\}$ differ

$\longrightarrow$ WL test decides they are non-isomorphic

$$\boldsymbol{h}_i^{(k)} = \boxed{\phi}\left(\boldsymbol{h}_i^{(k-1)}, \boxed{f}\left(\{\boldsymbol{h}_j^{(k-1)} : j \in \mathcal{N}(i)\}\right)\right)$$

Injective

03 Graph Isomorphism Network (GIN)

G, G' two non-isomorphic graphs

$\mathcal{A} : G \rightarrow \mathbb{R}^d$ a GNN

Construct $\mathcal{A}$ s.t. $\{h_i : i \in V(G)\}$ and $\{h_j : j \in V(G')\}$ differ

$\longrightarrow$ WL test decides they are non-isomorphic

$$\boldsymbol{h}_i^{(k)} = \boxed{\phi}\left(\boldsymbol{h}_i^{(k-1)}, \boxed{f}\left(\{\boldsymbol{h}_j^{(k-1)} : j \in \mathcal{N}(i)\}\right)\right)$$

Injective

Sum-decomposition

04 Sum-decomposition[3]

Any injective function on multisets can be decomposed as

$$g(X) = \phi\Big(\sum_{x \in X} f(x)\Big)$$

[3]Zaheer et al., *Deep sets*, Advances in Neural Information Processing Systems 30, 2017

04 Sum-decomposition[3]

Any injective function on multisets can be decomposed as

$$g(X) = \phi\left(\sum_{x \in X} f(x)\right)$$

$$g(\boldsymbol{h}, X) = \phi\Big((1+\epsilon) \cdot f(\boldsymbol{h}) + \sum_{\boldsymbol{x} \in X} f(\boldsymbol{x})\Big)$$

[3]Zaheer et al., *Deep sets*, Advances in Neural Information Processing Systems 30, 2017

04 Back to GIN

Use an MLP for representing $\phi \circ f$

$$\boldsymbol{h}_i^{(k)} = \mathrm{MLP}^{(k)}\left(\left(1 + \epsilon^{(k)}\right) \cdot \boldsymbol{h}_i^{(k-1)} + \sum_{j \in \mathcal{N}(i)} \boldsymbol{h}_j^{(k-1)}\right)$$

04 Back to GIN

Use an MLP for representing $\phi \circ f$

$$\boldsymbol{h}_i^{(k)} = \mathrm{MLP}^{(k)}\left(\left(1+\epsilon^{(k)}\right) \cdot \boldsymbol{h}_i^{(k-1)} + \sum_{j \in \mathcal{N}(i)} \boldsymbol{h}_j^{(k-1)}\right)$$

Cons of sum-decomposition:

- Highly discontinuous functions
- For uncountable domains, latent dimension of f should be higher than the number of elements in the set[4]
- No guarantee to find the right function

[4]Wagstaff et al., *On the limitations of representing functions on sets*, Proceedings of the 36th International Conference on Machine Learning, 2019

05 Principal Neighborhood Aggregation[5]

Select the best combination of aggregators and scalers

05 Principal Neighborhood Aggregation[5]

Select the best combination of aggregators and scalers

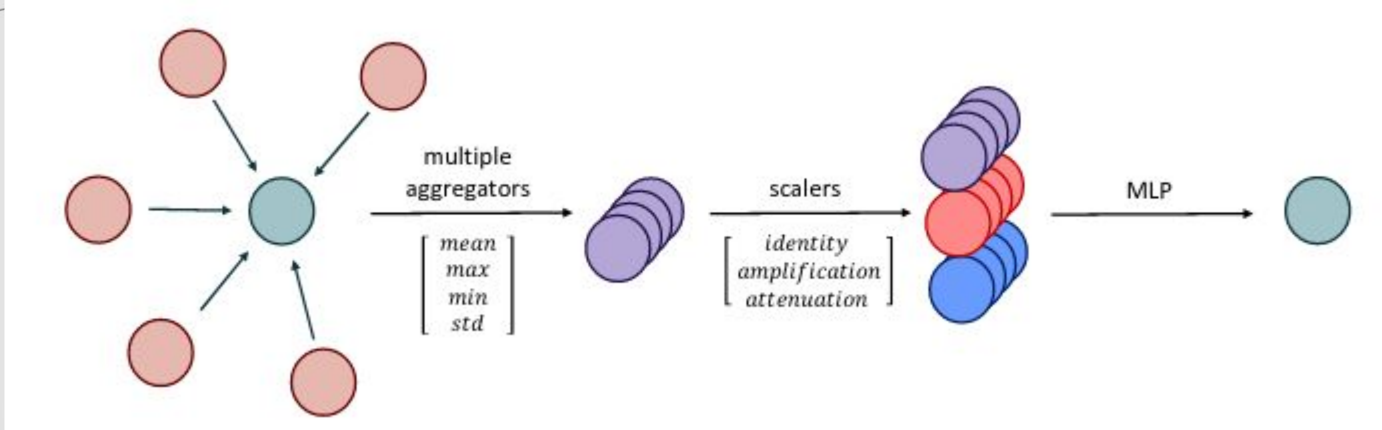


Image taken from the arXiv version of the paper.

[5]Corso et al., *Principal Neighbourhood Aggregation for Graph Nets*, Advances in Neural Information Processing Systems 33 (NeurIPS 2020), 2020

05 Principal Neighborhood Aggregation[5]

Select the best combination of aggregators and scalers

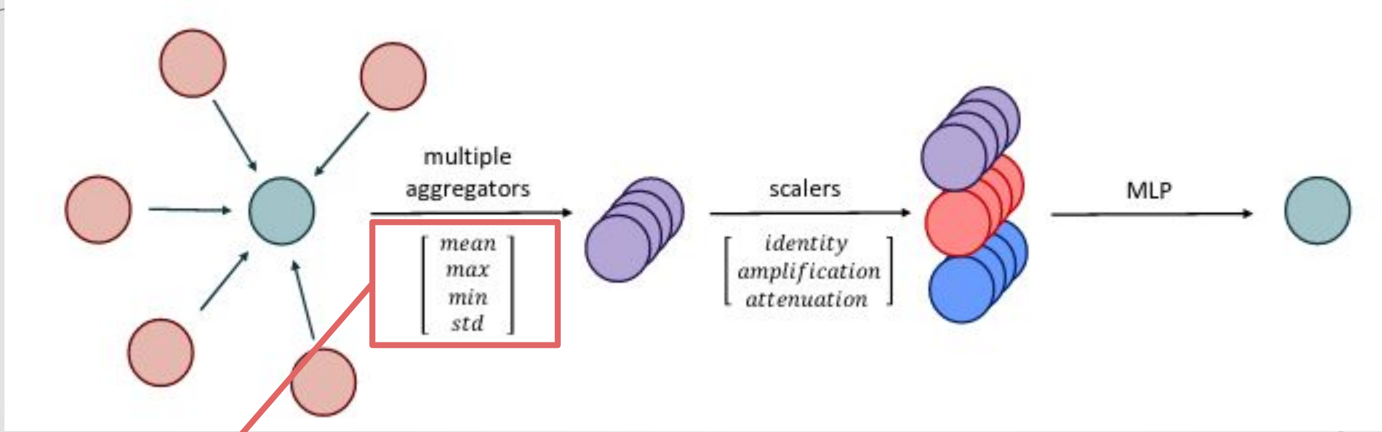


Image taken from the arXiv version of the paper.

Library of aggregators

[5]Corso et al., *Principal Neighbourhood Aggregation for Graph Nets*, Advances in Neural Information Processing Systems 33 (NeurIPS 2020), 2020

05 Principal Neighborhood Aggregation[5]

Select the best combination of aggregators and scalers

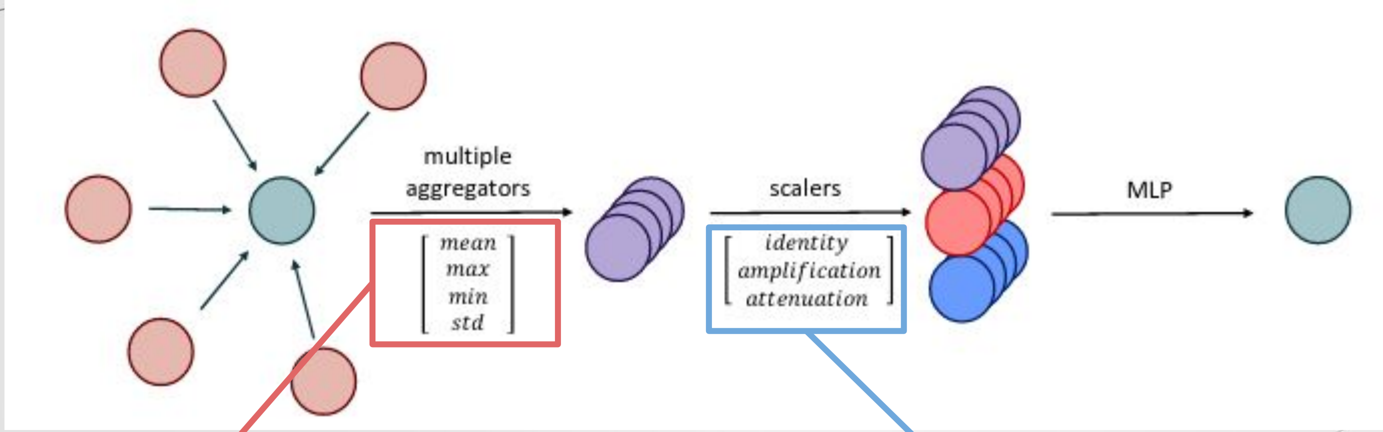


Image taken from the arXiv version of the paper.

Library of aggregators

Logarithmic scalers

[5]Corso et al., *Principal Neighbourhood Aggregation for Graph Nets*, Advances in Neural Information Processing Systems 33 (NeurIPS 2020), 2020

05 Principal Neighborhood Aggregation[5]

$$S = \left(\frac{\log(d+1)}{\delta}\right)^{\alpha}$$

$$\delta = \frac{1}{|train|} \sum_{i \in train} \log(d_i + 1)$$

$S_{amp}, \alpha = 1$ $S_{att}, \alpha = -1$ $S_{identity}$

06 Learning Aggregation Functions[6]

Don’t choose the aggregation function(s) - learn it!

[5]Pellegrini et al., *Learning Aggregation Functions*, under revision, 2020

06 Learning Aggregation Functions[6]

Don't choose the aggregation function(s) - learn it!

$$L_{a,b}(X) := \left(\sum_{x_i \in X} x_i^b \right)^a \qquad a, b \geq 0, x_i > 0$$

[5]Pellegrini et al., *Learning Aggregation Functions*, under revision, 2020

06 Learning Aggregation Functions[6]

Don't choose the aggregation function(s) - learn it!

$$L_{a,b}(X) := \left(\sum_{x_i \in X} x_i^b \right)^a \qquad a, b \geq 0, x_i > 0$$

$$LAF(\boldsymbol{X}) := \frac{\alpha L_{a,b}(\boldsymbol{X}) + \beta L_{c,d}(\mathbf{1} - \boldsymbol{X})}{\gamma L_{e,f}(\boldsymbol{X}) + \delta L_{g,h}(\mathbf{1} - \boldsymbol{X})}$$

[5]Pellegrini et al., *Learning Aggregation Functions*, under revision, 2020

06 Learning Aggregation Functions[6]

Don't choose the aggregation function(s) - learn it!

$$L_{a,b}(X) := \left(\sum_{x_i \in X} x_i^{\boxed{b}} \right)^{\boxed{a}} \qquad a, b \geq 0, x_i > 0$$

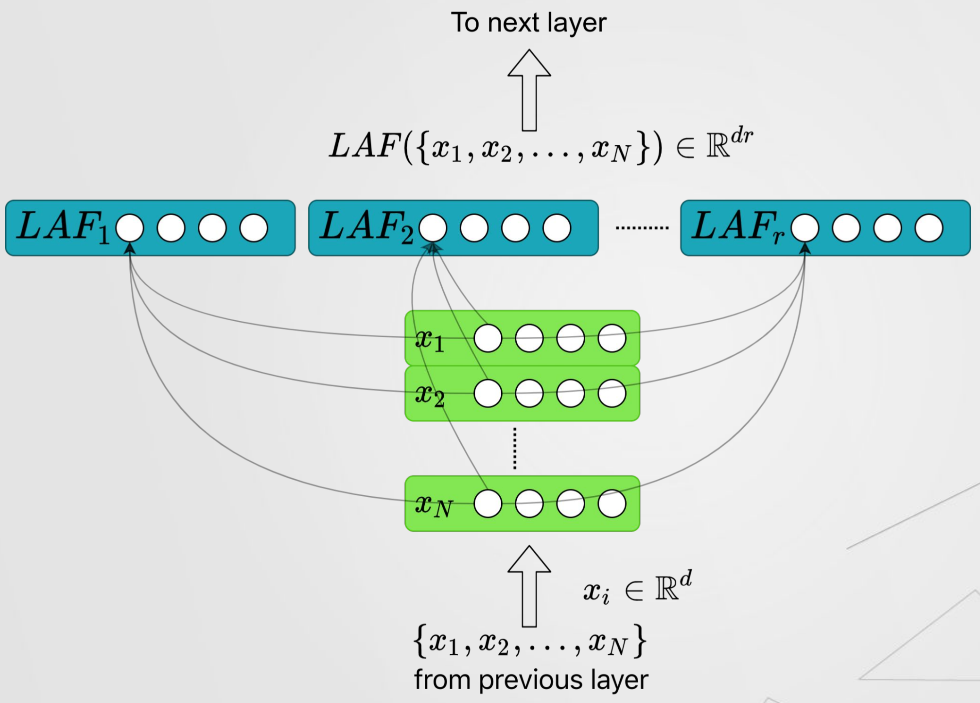
Learnable parameters

$$LAF(\boldsymbol{X}) := \frac{\alpha L_{a,b}(\boldsymbol{X}) + \boxed{\beta} L_{c,d}(\boldsymbol{1} - \boldsymbol{X})}{\gamma L_{e,f}(\boldsymbol{X}) + \delta L_{g,h}(\boldsymbol{1} - \boldsymbol{X})}$$

MAX, MIN, SUM, MEAN, MOMENTS, MIN/MAX, COUNT ...

[5]Pellegrini et al., *Learning Aggregation Functions*, under revision, 2020

06 Learning Aggregation Functions



07 Aggregation in Pytorch Geometric

PyTorch Geometric provides the MessagePassing base class.

METHODS

CLASS **MessagePassing** (**aggr**: Optional[str] = 'add', **flow**: str = 'source_to_target', **node_dim**: int = - 2) [source]

Aggregates messages from neighbors (sum, mean, max)

aggregate (**inputs**: torch.Tensor, **index**: torch.Tensor, **ptr**: Optional[torch.Tensor] = None, **dim_size**: Optional[int] = None) → **torch.Tensor** [source]

Constructs messages from node j to node i in analogy to ϕΘ

message (**x_j**: torch.Tensor) → **torch.Tensor** [source]

Propagate messages

propagate (**edge_index**: Union[torch.Tensor, torch_sparse.tensor.SparseTensor], **size**: Optional[Tuple[int, int]] = None, ****kwargs**) [source]

Updates node embeddings in analogy to γΘ

update (**inputs**: torch.Tensor) → **torch.Tensor** [source]

07 Aggregation in Pytorch Geometric

PyTorch Geometric provides the MessagePassing base class.

METHODS

CLASS **MessagePassing** (**aggr**: Optional[str] = 'add', **flow**: str = 'source_to_target', **node_dim**: int = - 2) [source]

Aggregates messages from neighbors (sum, mean, max)

aggregate (**inputs**: torch.Tensor, **index**: torch.Tensor, **ptr**: Optional[torch.Tensor] = None, **dim_size**: Optional[int] = None) → **torch.Tensor** [source]

Constructs messages from node j to node i in analogy to ϕΘ

message (**x_j**: torch.Tensor) → **torch.Tensor** [source]

Propagate messages

propagate (**edge_index**: Union[torch.Tensor, torch_sparse.tensor.SparseTensor], **size**: Optional[Tuple[int, int]] = None, ****kwargs**) [source]

Updates node embeddings in analogy to γΘ

update (**inputs**: torch.Tensor) → **torch.Tensor** [source]