Inductive Representation Learning on Large Graph

William L. Hamilton, Rex Ying and Jure Leskovec

Stanford University

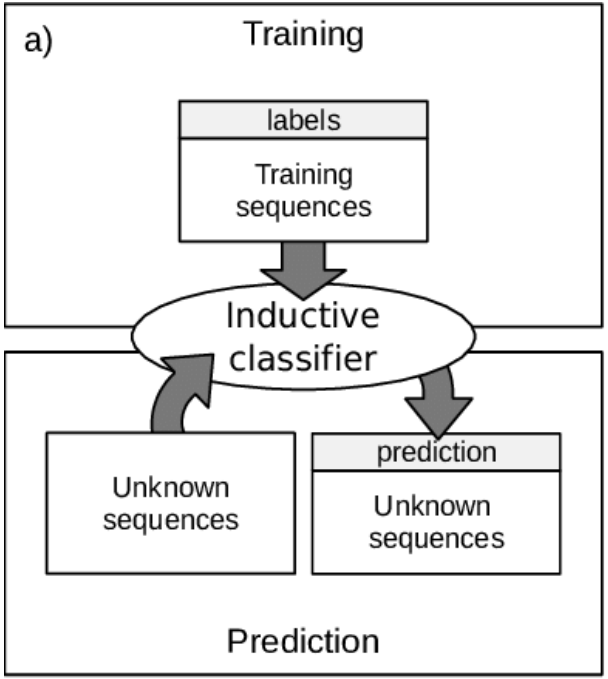
NIPS 2017

서울시립대학교
UNIVERSITY OF SEOUL

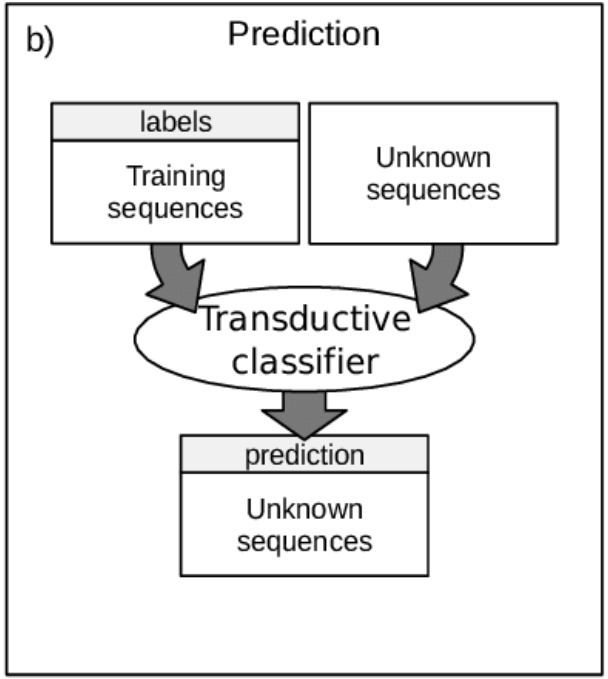
cheon.research@gmail.com

2020. 09. 22.

Inductive Learning



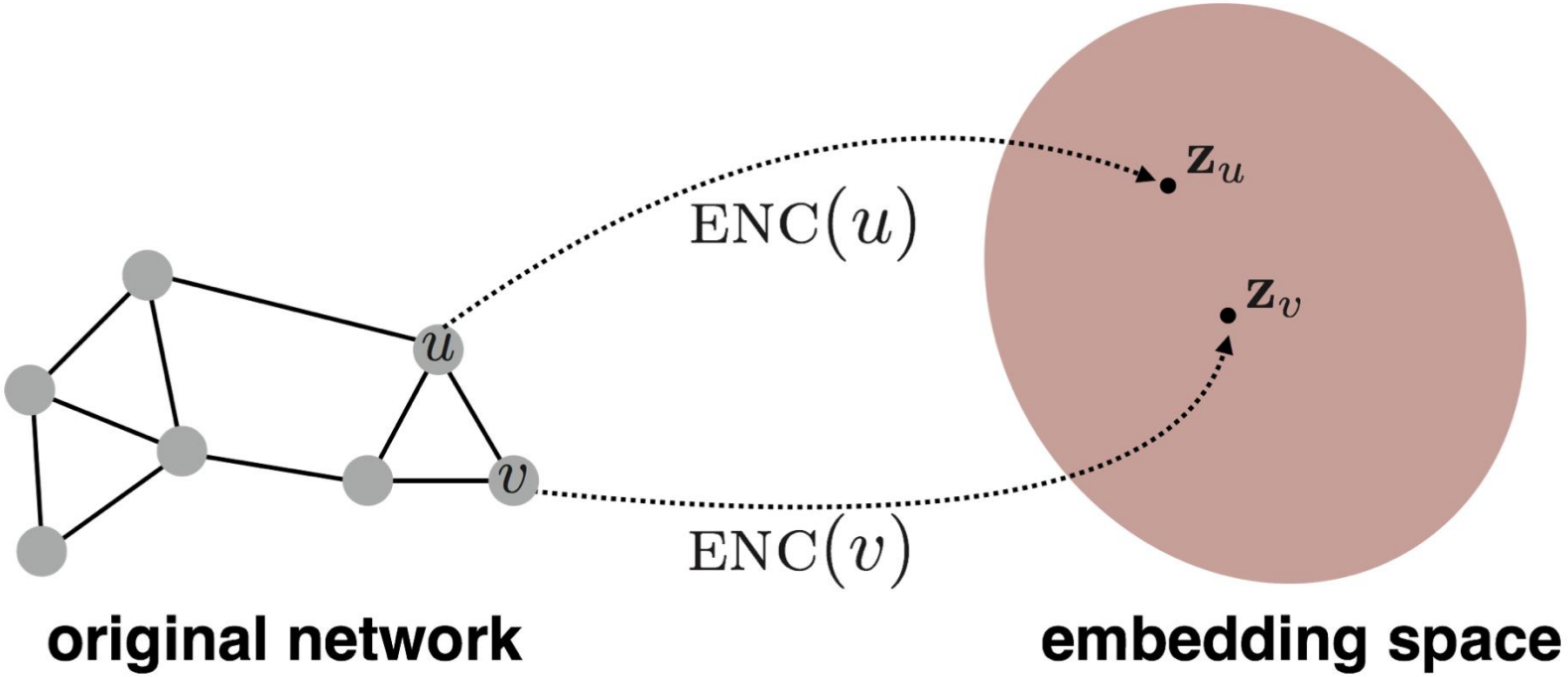
Transductive Learning



https://academic.oup.com/bioinformatics/article/34/4/541/4222633

서울시립대학교 UNIVERSITY OF SEOUL

Node Embedding



Factorization-based Embedding

- Methods learn low-dimensional embeddings using random walk statistics and matrix factorization-based learning objectives.

- These algorithms directly train node embeddings for individual nodes.

- The objective function is invariant to orthogonal transformations.

→ The embedding space does not naturally generalize between graphs and can drift during re-training.

Supervised Learning over Graphs

- Previous approaches attempt to classify entire graphs or subgraphs.

Graph Convolutional Networks

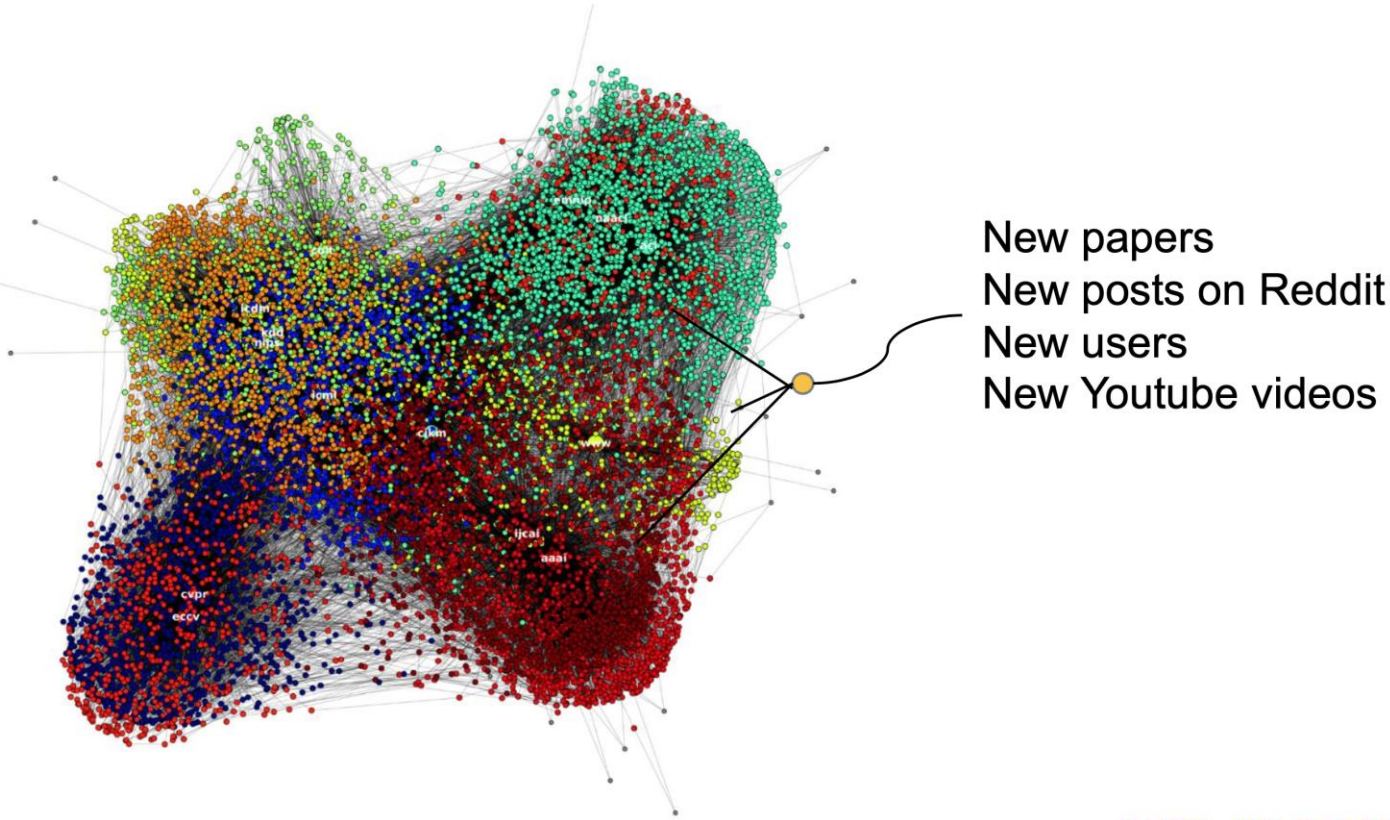
- The methods do not scale to large graphs or are designed for graph classification.
- GCN algorithm is designed for semi-supervised learning in a transductive setting.

Previous Works

- Previous works have focused on embedding nodes from a single fixed graph.
- Many real-world applications require embeddings to be quickly generated for unseen nodes, or entirely new graphs or subgraphs.
- Most existing approaches to generating node embeddings are inherently transductive.
- Graph Convolutional Networks(GCNs) have only been applied in the transductive setting with fixed graph.

Problems

- ✓ Transductive
- ✓ Full Batch
- ✓ Memory on Large Graph



Inductive Representation Learning on Large Graphs

William L. Hamilton* wleif@stanford.edu

Rex Ying* rexying@stanford.edu

Jure Leskovec jure@cs.stanford.edu

Department of Computer Science
Stanford University
Stanford, CA, 94305

GraphSAGE

GraphSAGE(**SA**mple and aggre**G**at**E**) for inductive node embedding.

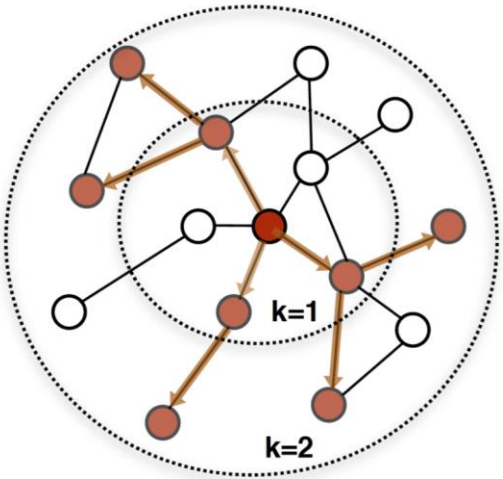
- Leveraging node features in order to learn an embedding function that generalizes to unseen nodes.

  (also be applied to graphs without node features.)

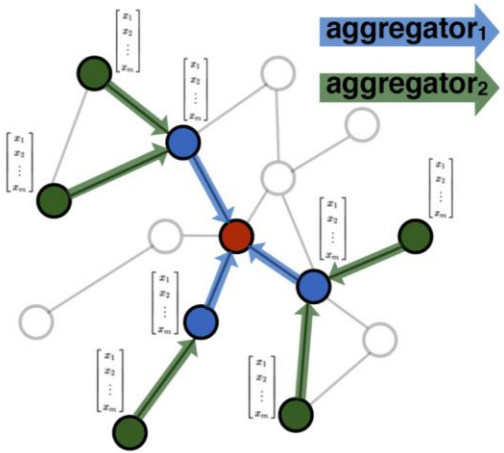
- Simultaneously learn the topological structure of each node's neighborhood as well as the distribution of node features in the neighborhood.

GraphSAGE

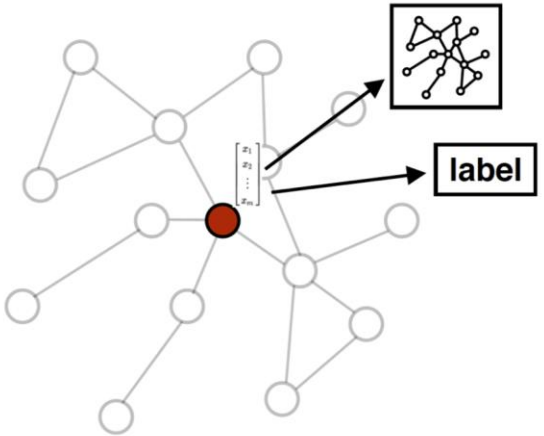
*How to **aggregate** feature information from a node's local neighborhood*



1. Sample neighborhood



2. Aggregate feature information from neighbors



3. Predict graph context and label using aggregated information

Embedding Generation Algorithm

After the model has already been trained and the parameters are fixed.

Algorithm 1: GraphSAGE embedding generation (i.e., forward propagation) algorithm

Input : Graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$; input features $\{\mathbf{x}_v, \forall v \in \mathcal{V}\}$; depth K; weight matrices $\mathbf{W}^k, \forall k \in \{1, ..., K\}$; non-linearity σ; differentiable aggregator functions $\text{AGGREGATE}_k, \forall k \in \{1, ..., K\}$; neighborhood function $\mathcal{N} : v \rightarrow 2^{\mathcal{V}}$

Output : Vector representations $\mathbf{z}_v$ for all $v \in \mathcal{V}$

$\mathbf{h}_v^0 \leftarrow \mathbf{x}_v, \forall v \in \mathcal{V}$;
for $k = 1...K$ **do**
  for $v \in \mathcal{V}$ **do**
    $\mathbf{h}_{\mathcal{N}(v)}^k \leftarrow \text{AGGREGATE}_k(\{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\})$;
    $\mathbf{h}_v^k \leftarrow \sigma\left(\mathbf{W}^k \cdot \text{CONCAT}(\mathbf{h}_v^{k-1}, \mathbf{h}_{\mathcal{N}(v)}^k)\right)$
  end
  $\mathbf{h}_v^k \leftarrow \mathbf{h}_v^k / \|\mathbf{h}_v^k\|_2, \forall v \in \mathcal{V}$
end
$\mathbf{z}_v \leftarrow \mathbf{h}_v^K, \forall v \in \mathcal{V}$

Algorithm 2: GraphSAGE minibatch forward propagation algorithm

Input : Graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$;
input features $\{\mathbf{x}_v, \forall v \in \mathcal{B}\}$;
depth K; weight matrices $\mathbf{W}^k, \forall k \in \{1, ..., K\}$;
non-linearity σ;
differentiable aggregator functions $\text{AGGREGATE}_k, \forall k \in \{1, ..., K\}$;
neighborhood sampling functions, $\mathcal{N}_k : v \rightarrow 2^{\mathcal{V}}, \forall k \in \{1, ..., K\}$

Output : Vector representations $\mathbf{z}_v$ for all $v \in \mathcal{B}$

$\mathcal{B}^K \leftarrow \mathcal{B}$;
for $k = K...1$ **do**
  $B^{k-1} \leftarrow \mathcal{B}^k$;
  for $u \in \mathcal{B}^k$ **do**
    $\mathcal{B}^{k-1} \leftarrow \mathcal{B}^{k-1} \cup \mathcal{N}_k(u)$;
  end
end
$\mathbf{h}_u^0 \leftarrow \mathbf{x}_v, \forall v \in \mathcal{B}^0$;
for $k = 1...K$ **do**
  for $u \in \mathcal{B}^k$ **do**
    $\mathbf{h}_{\mathcal{N}(u)}^k \leftarrow \text{AGGREGATE}_k(\{\mathbf{h}_{u'}^{k-1}, \forall u' \in \mathcal{N}_k(u)\})$;
    $\mathbf{h}_u^k \leftarrow \sigma\left(\mathbf{W}^k \cdot \text{CONCAT}(\mathbf{h}_u^{k-1}, \mathbf{h}_{\mathcal{N}(u)}^k)\right)$;
    $\mathbf{h}_u^k \leftarrow \mathbf{h}_u^k / \|\mathbf{h}_u^k\|_2$;
  end
end
$\mathbf{z}_u \leftarrow \mathbf{h}_u^K, \forall u \in \mathcal{B}$

Aggregator Architectures

- Mean Aggregator

$$\mathbf{h}_v^k \leftarrow \sigma(\mathbf{W} \cdot \text{MEAN}(\{\mathbf{h}_v^{k-1}\} \cup \{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\})$$

- LSTM Aggregator

$$\text{LSTM}\left([\mathbf{h}_u^{k-1}, \forall u \in \pi(N(v))]\right)$$

- Pooling Aggregator

$$\text{AGGREGATE}_k^{\text{pool}} = \max(\{\sigma\left(\mathbf{W}_{\text{pool}}\mathbf{h}_{u_i}^k + \mathbf{b}\right), \forall u_i \in \mathcal{N}(v)\})$$

Neighborhood Definition

Uniformly sample a fixed-size set of neighbors instead of using full neighborhoods sets. (drawing different uniform samples at each iteration)

Practically, GraphSAGE could achieve high performance with

- ✓ neighborhood $K = 2$
- ✓ samples $S_1 \cdot S_2 \leq 500$

Learning the Parameters of GrahpSAGE

To encourage nearby nodes to have **similar** representations, while enforcing that the representations of disparate nodes are highly **distinct**.

output representations

number of negative samples

$$J_{\mathcal{G}}(\mathbf{z}_u) = -\log\left(\sigma(\mathbf{z}_u^\top \mathbf{z}_v)\right) - Q \cdot \mathbb{E}_{v_n \sim P_n(v)} \log\left(\sigma(-\mathbf{z}_u^\top \mathbf{z}_{v_n})\right)$$

neighbor node

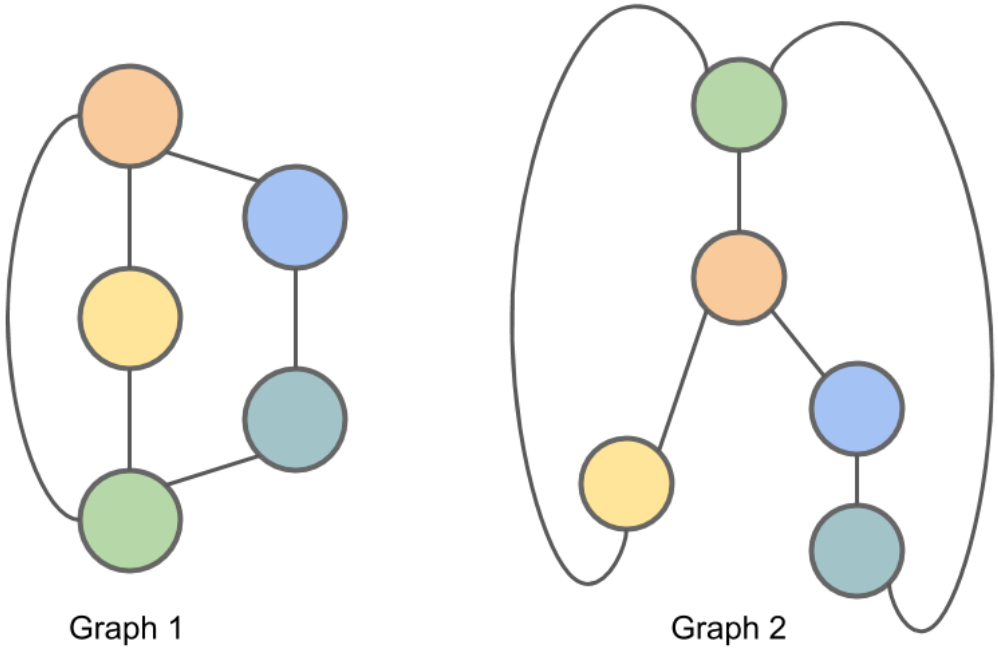
Relation to the Isomorphism Test

The GraphSAGE is conceptually inspired by a testing graph isomorphism.

Weisfeiler-Leman

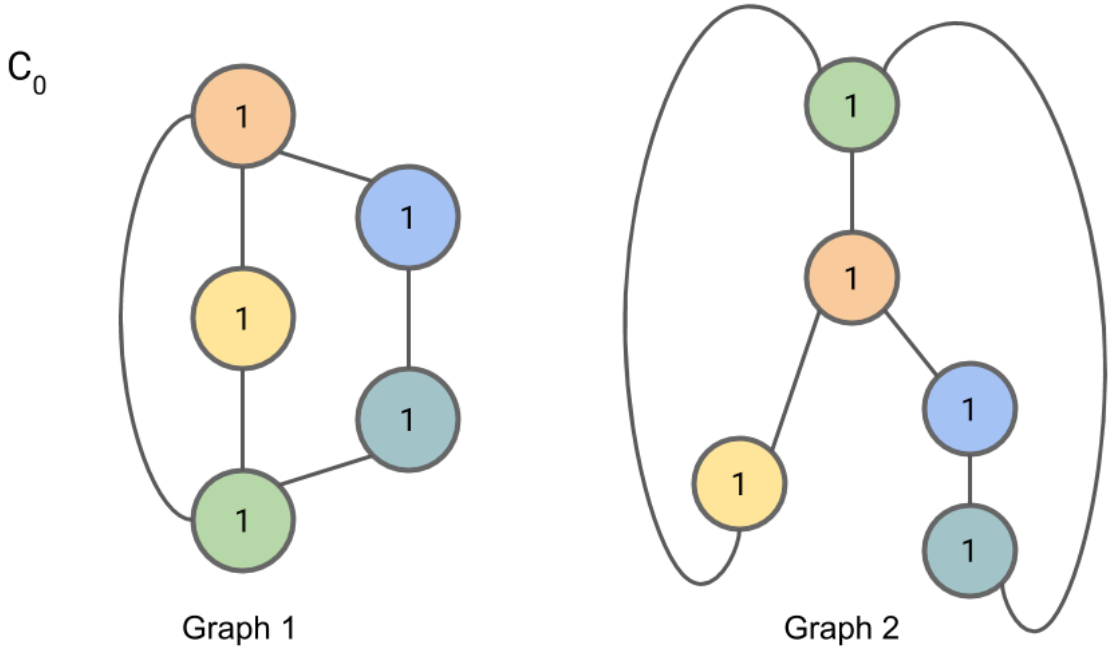
Isomorphism Test

WL Isomorphism Test (1)



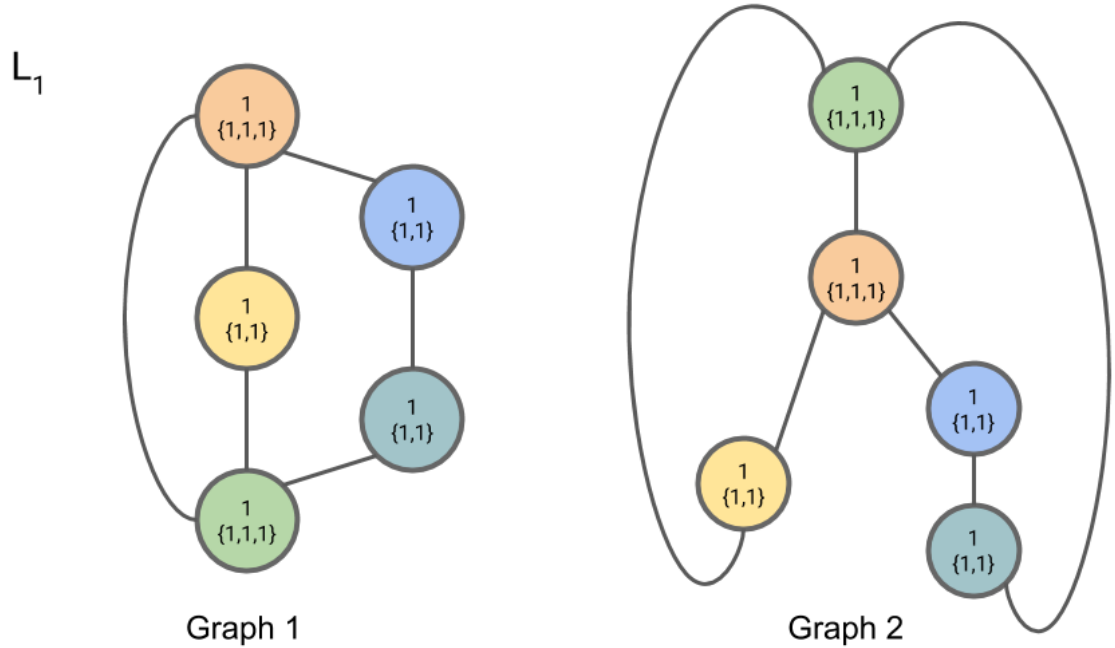
https://davidbieber.com/post/2019-05-10-weisfeiler-lehman-isomorphism-test/

WL Isomorphism Test (2)



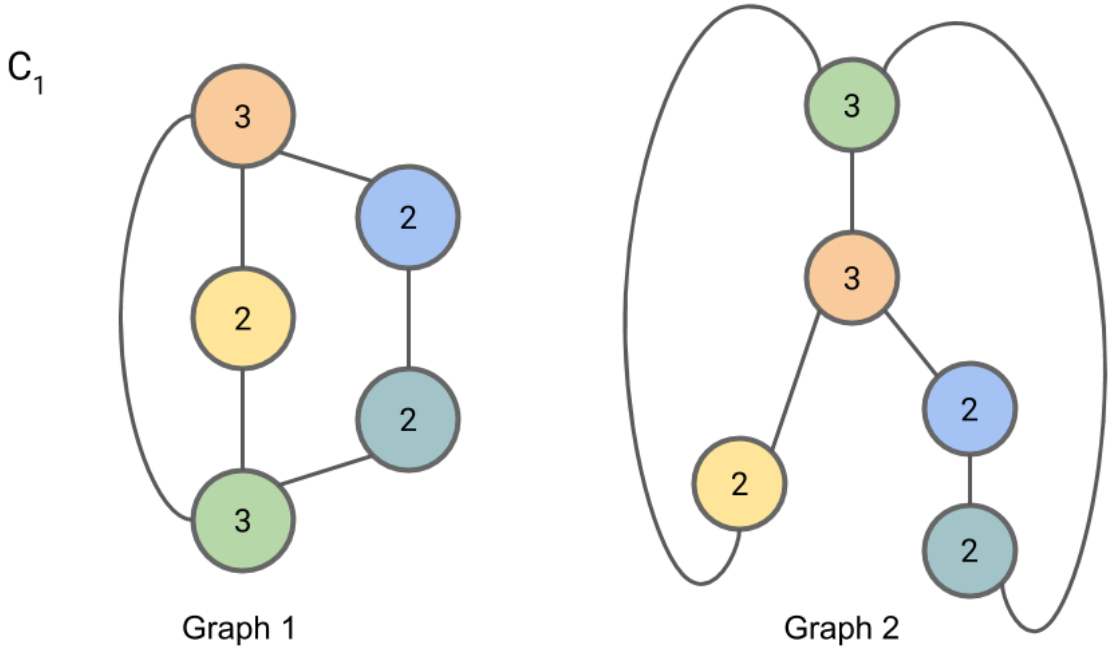
https://davidbieber.com/post/2019-05-10-weisfeiler-lehman-isomorphism-test/

WL Isomorphism Test (3)



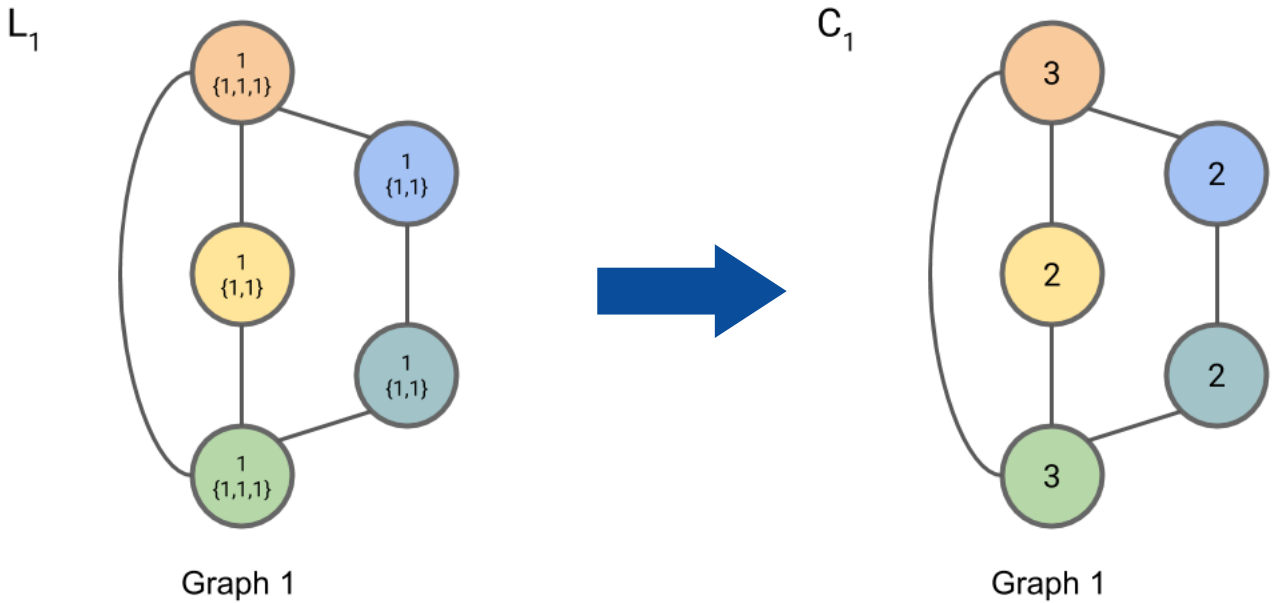
https://davidbieber.com/post/2019-05-10-weisfeiler-lehman-isomorphism-test/

WL Isomorphism Test (4)



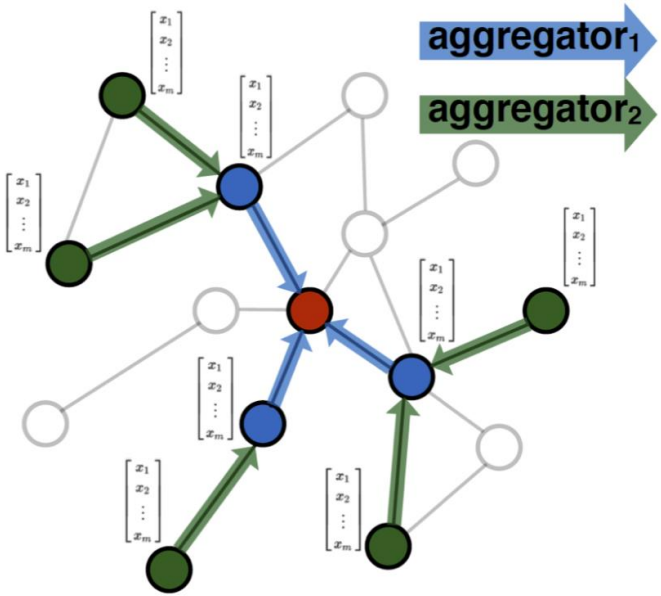
Relation to the Isomorphism Test

The GraphSAGE is conceptually inspired by a testing graph isomorphism.

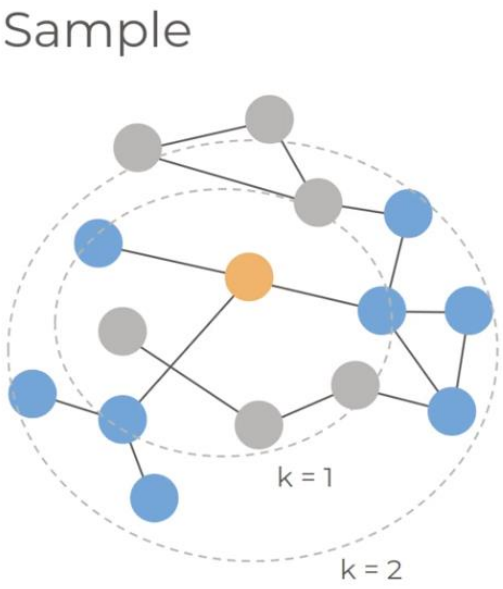


Relation to the Isomorphism Test

Replacing with hash function with **trainable neural network aggregator**.



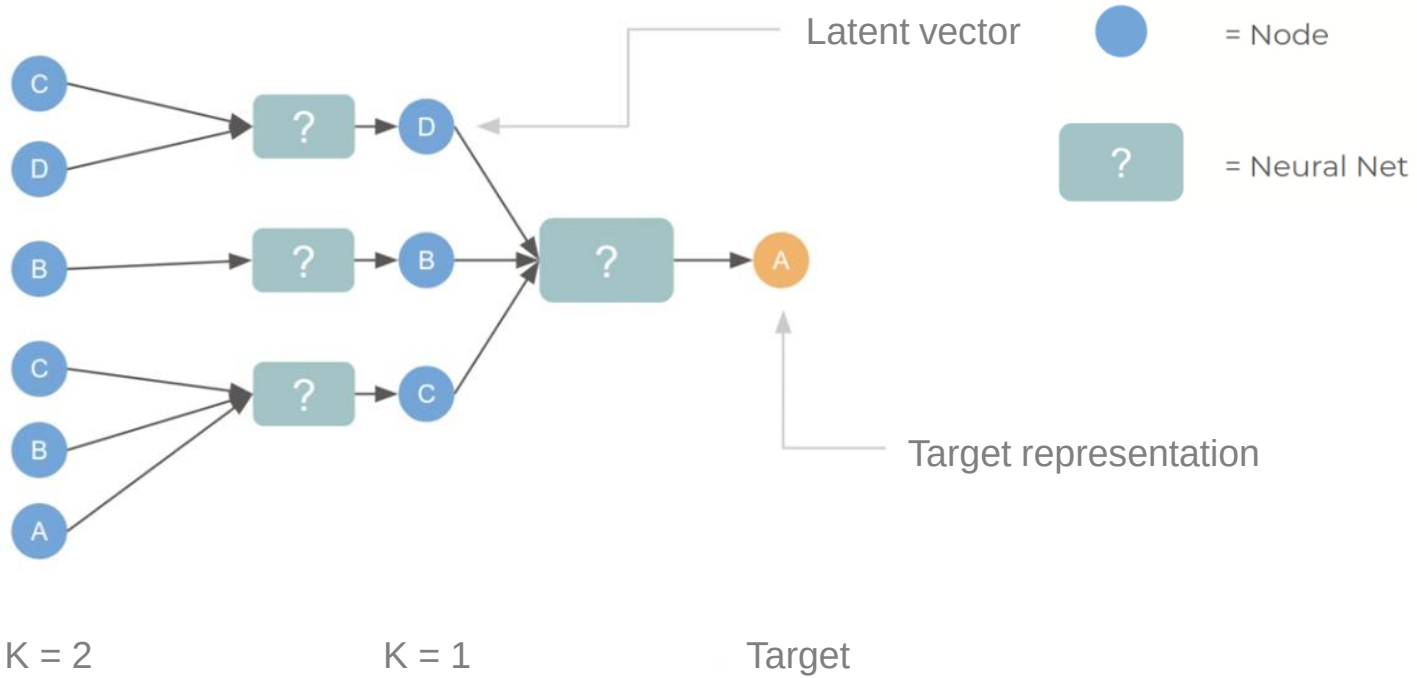
GraphSAGE



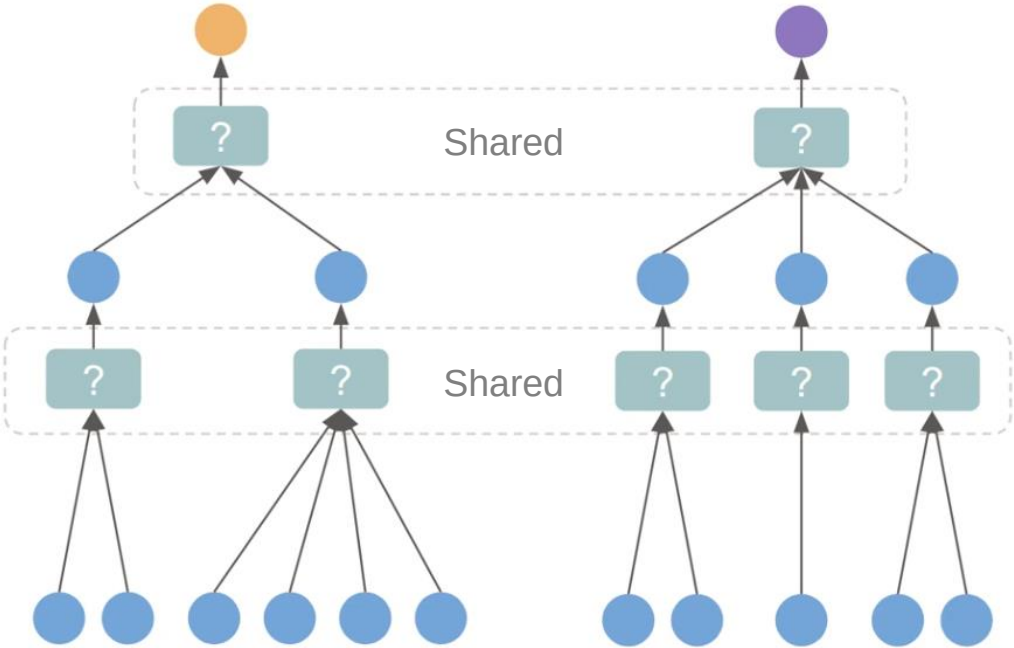
https://www.youtube.com/watch?v=rGn3bmoxnJU&ab_channel=Scarlett%27sLog

서울시립대학교 UNIVERSITY OF SEOUL

GraphSAGE


Latent vector
= Node
?
= Neural Net
C
D
B
C
B
A
?
?
?
D
B
C
A
Target representation
K = 2
K = 1
Target

GraphSAGE



https://www.youtube.com/watch?v=rGn3bmoxnJU&ab_channel=Scarlett%27sLog

서울시립대학교 UNIVERSITY OF SEOUL

Experiments

- Datasets

  Academic papers / Reddit posts / Protein-Protein Interactions graphs

  (predictions on nodes or graphs that **are not seen during training**)

- Experiment Goals

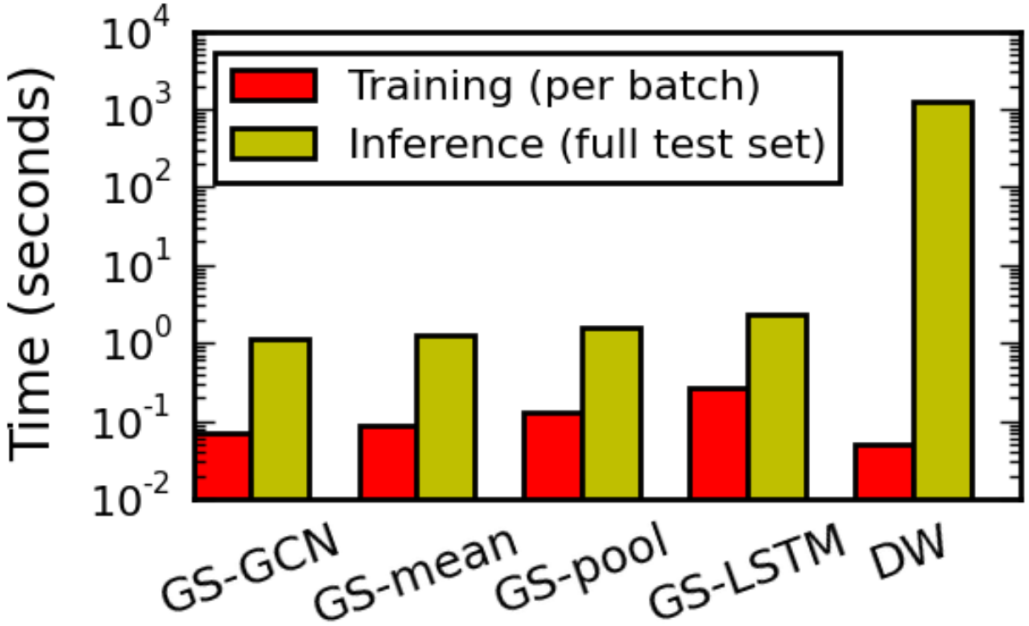
  1) Verifying the improvement of GraphSAGE over the baselines.
  2) Providing a rigorous comparison of the different aggregators.

Prediction Results

Table 1: Prediction results for the three datasets (micro-averaged F1 scores). Results for unsupervised and fully supervised GraphSAGE are shown. Analogous trends hold for macro-averaged scores.

	Citation		Reddit		PPI	
Name	Unsup. F1	Sup. F1	Unsup. F1	Sup. F1	Unsup. F1	Sup. F1
Random	0.206	0.206	0.043	0.042	0.396	0.396
Raw features	0.575	0.575	0.585	0.585	0.422	0.422
DeepWalk	0.565	0.565	0.324	0.324	—	—
DeepWalk + features	0.701	0.701	0.691	0.691	—	—
GraphSAGE-GCN	0.742	0.772	**0.908**	0.930	0.465	0.500
GraphSAGE-mean	0.778	0.820	0.897	0.950	0.486	0.598
GraphSAGE-LSTM	0.788	0.832	**0.907**	**0.954**	0.482	**0.612**
GraphSAGE-pool	**0.798**	**0.839**	0.892	0.948	**0.502**	0.600
% gain over feat.	39%	46%	55%	63%	19%	45%

Timing Experiments on Reddit Data



Model Performance on Citation Data