



# **Paper Review**

# **Inductive Representation Learning on Large Graphs**

(GraphSAGE)

**William L. Hamilton, Rex Ying, Jure Leskovec**

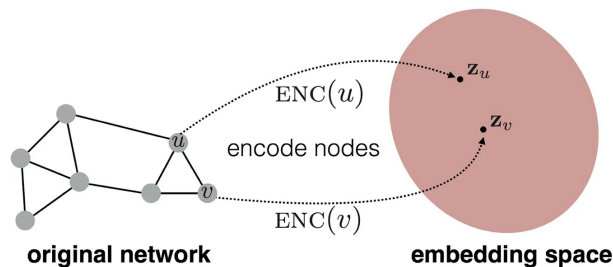
**2017**

**Data Mining And Intelligence Systems (DMAIS) Lab**  
**School of Computer Science and Engineering**  
**Chung-Ang University**

- ☐ Introduction
- ☐ Problem
- ☐ Present Work
- ☐ Proposed Method
- ☐ Experiments
- ☐ Conclusion

## □ Low-dimensional vector embeddings of nodes

- In large graphs, they have proved extremely useful in a wide variety of prediction and graph analysis tasks.
- The basic idea is to use dimensionality reduction techniques
  - From the high-dimensional information about a node's neighborhood into a dense vector embeddings
  - To be fed to downstream machine learning systems to perform various tasks such as node classification, clustering, and link prediction.

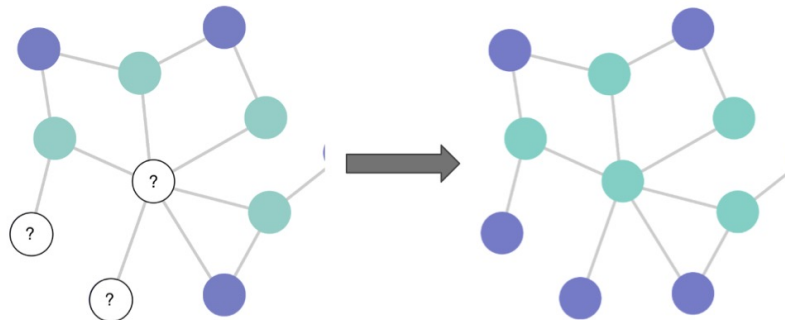


Mapping Process in Graph Representation Learning (src: [stanford-cs224w](https://stanford-cs224w))

# Two different paradigms for learning on graphs

## □ (1) Transductive Learning

- Node and edge sets remain constant across the training and prediction phases.
- It does not support generalization to unseen nodes and edges.

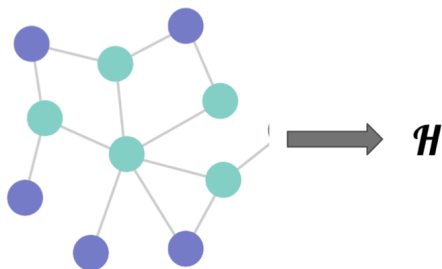


**Figure 1: Node classification in transductive settings.** At the training time, the learning algorithm has access to all the nodes and edges, including those nodes for which labels are to be predicted (denoted by question marks).

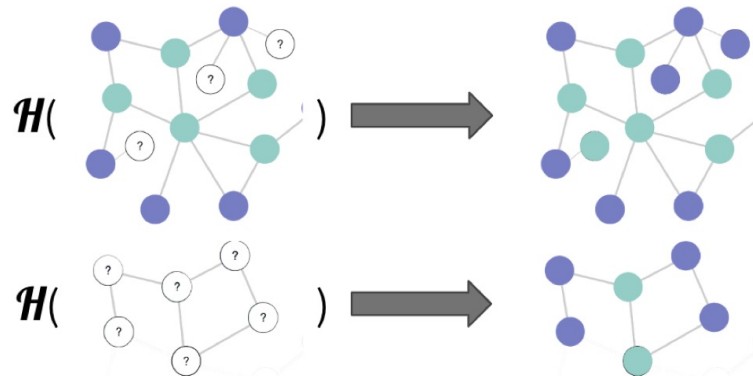
# Two different paradigms for learning on graphs

## □ Inductive Learning

- First, a model is learned over the training graph consisting of some nodes and edges.
- The learned model is then used to predict on unseen nodes and edges
  - The prediction ones may or may not be disconnected from the nodes and edges in the training graph.



(a) A model  $\mathcal{H}$  is learned over some graph

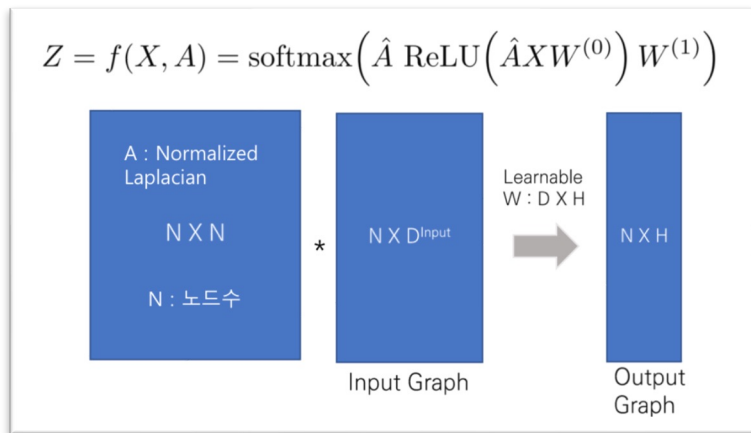


(b) The model is then by applied to new nodes and edges

- **Previous Works are in transductive setting**
  - Focused on embedding nodes from a **single fixed graph**
  - All nodes in the graph are required during training
- **Real-world applications require embeddings for **unseen nodes**, or **entirely new (sub)graphs** -> *inductive***
  - Generalization across graphs with the same form of features is facilitated
- **Inductive node embedding problem is especially difficult**
  - Because generalizing to unseen nodes requires “aligning” newly observed subgraphs to the node embeddings that the algorithm has already optimized on
  - To recognize structural properties of a node’s neighborhood
    - Node’s local role in the graph, as well as its global position

## □ Graph Convolution Network (GCN)

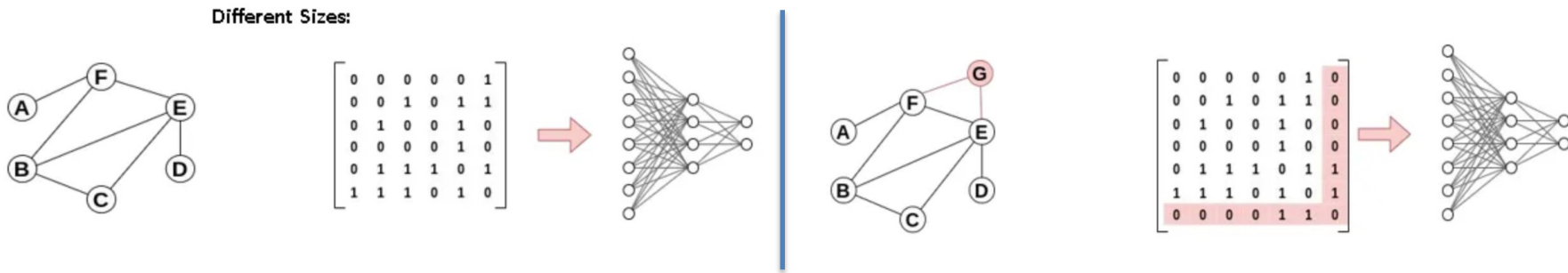
- Node embeddings in *transductive* setting
- Directly optimize the embeddings by using *matrix-factorization-based* objectives
  - Make predictions on nodes in a *single, fixed graph*
  - Not naturally generalize to unseen data



## □ Graph Convolution Network (GCN)

■ Modified to operate in an inductive settings

- Computationally expensive
- Requiring additional rounds of gradient descent before new prediction can be made





# Proposed Method

- GraphSAGE (SAmple and aggreGatE) for inductive node embedding
- Key idea : aggregating feature information from a node's local neighborhood

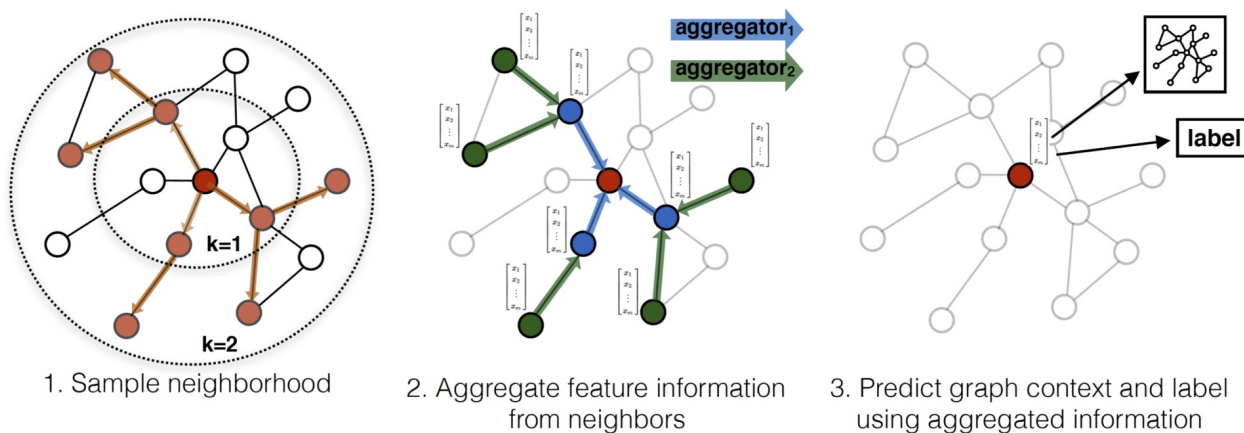


Figure 1: Visual illustration of the GraphSAGE sample and aggregate approach.

## ☐ GraphSAGE (SAmple and aggreGatE) for inductive node embedding

### ■ Feature Incorporation

- ☐ Leveraging node features (e.g., text attributes, profile information, node degrees)
- ☐ To learn an embedding function that generalizes to unseen nodes

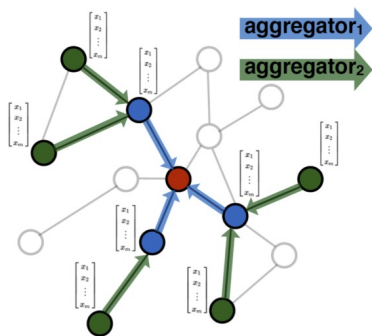
### ■ By incorporating node features in learning algorithm, It can learn

- ☐ The topological structure of each node's neighborhood
- ☐ The distribution of node features in the neighborhood

## □ GraphSAGE (SAmple and aggreGatE) for inductive node embedding

### ■ Aggregation Functions

- Instead of training a distinct embedding vector for each node, a set of aggregator functions is trained
  - Learning to aggregate features information from a node's local neighborhood



2. Aggregate feature information from neighbors

□ Assuming that the GraphSAGE model parameters are already learned

---

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

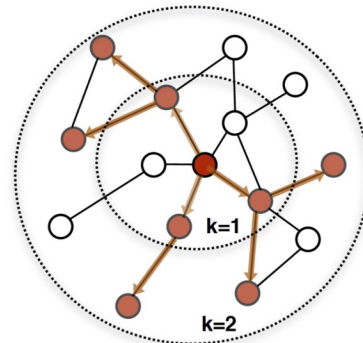
---

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

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

```
1  $\mathbf{h}_v^0 \leftarrow \mathbf{x}_v, \forall v \in \mathcal{V}$ ;  
2 for  $k = 1 \dots K$  do  
3   for  $v \in \mathcal{V}$  do  
4      $\mathbf{h}_{\mathcal{N}(v)}^k \leftarrow \text{AGGREGATE}_k(\{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\})$ ;  
5      $\mathbf{h}_v^k \leftarrow \sigma(\mathbf{W}^k \cdot \text{CONCAT}(\mathbf{h}_v^{k-1}, \mathbf{h}_{\mathcal{N}(v)}^k))$   
6   end  
7    $\mathbf{h}_v^k \leftarrow \mathbf{h}_v^k / \|\mathbf{h}_v^k\|_2, \forall v \in \mathcal{V}$   
8 end  
9  $\mathbf{z}_v \leftarrow \mathbf{h}_v^K, \forall v \in \mathcal{V}$ 
```

---

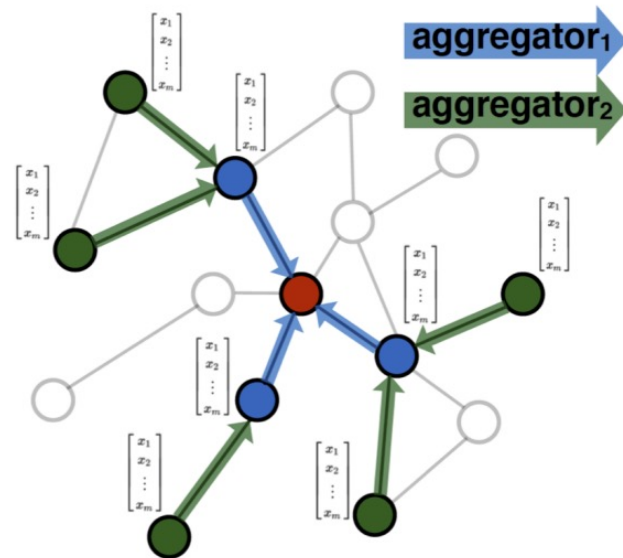


1. Sample neighborhood

# GraphSAGE Algorithm

```
1  $\mathbf{h}_v^0 \leftarrow \mathbf{x}_v, \forall v \in \mathcal{V}$ ;  
2 for  $k = 1 \dots K$  do  
3   for  $v \in \mathcal{V}$  do  
4      $\mathbf{h}_{\mathcal{N}(v)}^k \leftarrow \text{AGGREGATE}_k(\{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\})$ ;  
5      $\mathbf{h}_v^k \leftarrow \sigma(\mathbf{W}^k \cdot \text{CONCAT}(\mathbf{h}_v^{k-1}, \mathbf{h}_{\mathcal{N}(v)}^k))$   
6   end  
7    $\mathbf{h}_v^k \leftarrow \mathbf{h}_v^k / \|\mathbf{h}_v^k\|_2, \forall v \in \mathcal{V}$   
8 end  
9  $\mathbf{z}_v \leftarrow \mathbf{h}_v^K, \forall v \in \mathcal{V}$ 
```

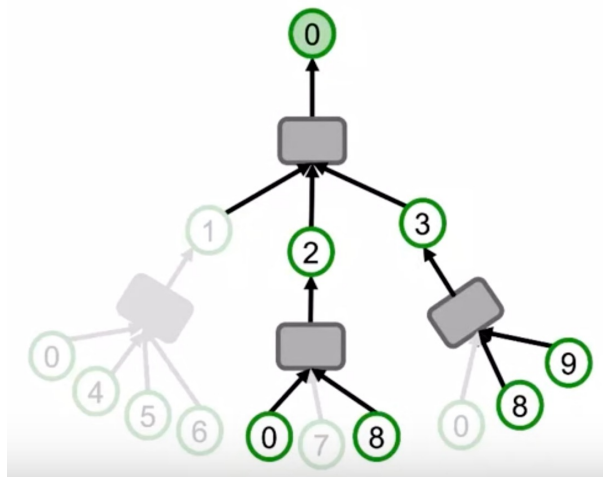
p.s. Handling unordered sets



2. Aggregate feature information from neighbors

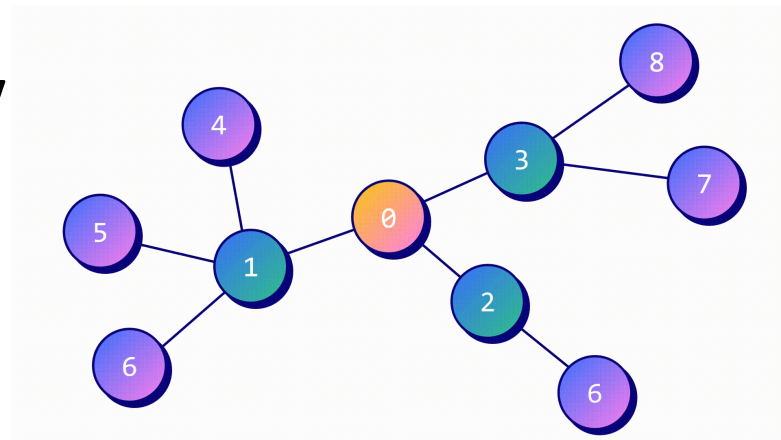
# Node Sampling

- Mini Batch is considered only a **fixed-sized set** of neighbors
  - Define
    - the **number of neighbors** ( $H = 2$ , hub nodes)
    - the number of neighbors of neighbors or **search depths** (k-hop neighborhood)
  - Uniform random draw in every iteration



# Node Sampling

- Keep the computational footprint of each batch fixed
- Balancing local and global information
- Without this sampling
  - Memory and expected runtime of a single batch is **unpredictable**
  - In worst case,  $O(|V|)$
- **Per-batch space and Time complexity**
  - Fixed :  $O(\prod_{i=1}^K S_i)$

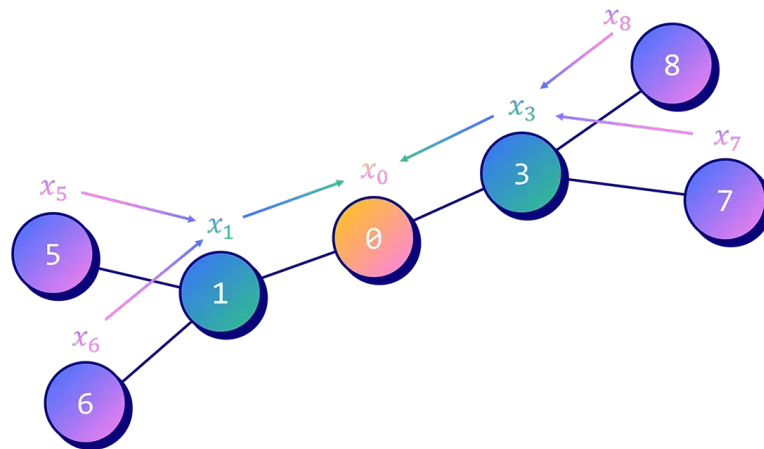


# Aggregator function

□ How to combine the feature vectors to produce the node embeddings

- Mean aggregator
- LSTM aggregator
- Pooling aggregator

```
3  for  $v \in \mathcal{V}$  do
4     $\mathbf{h}_{\mathcal{N}(v)}^k \leftarrow \text{AGGREGATE}_k(\{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\})$ 
5     $\mathbf{h}_v^k \leftarrow \sigma(\mathbf{W}^k \cdot \text{CONCAT}(\mathbf{h}_v^{k-1}, \mathbf{h}_{\mathcal{N}(v)}^k))$ 
6  end
```

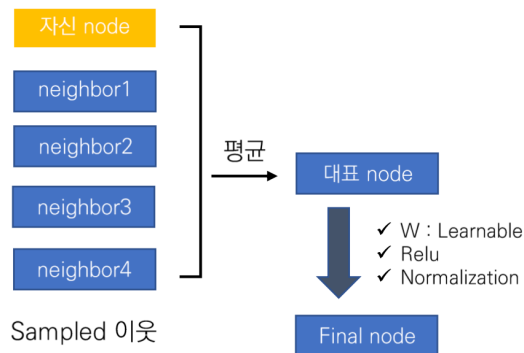




# Aggregator function (1)

## □ Mean aggregator

- Closed to GCN approach
- Average of the hidden features of the target node and its selected neighbors
- Simplest but less expressive

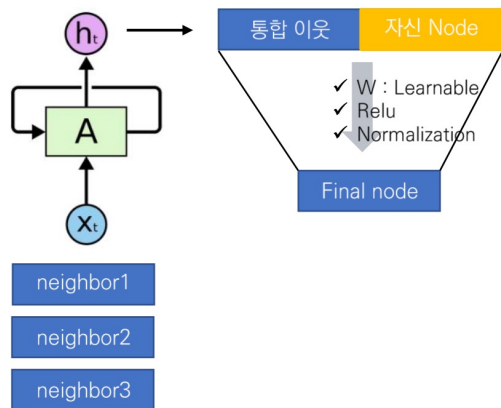


$$\mathbf{h}_v^k \leftarrow \sigma(\mathbf{W} \cdot \text{MEAN}(\{\mathbf{h}_v^{k-1}\} \cup \{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\})).$$

# Aggregator function (2)

## □ LSTM aggregator

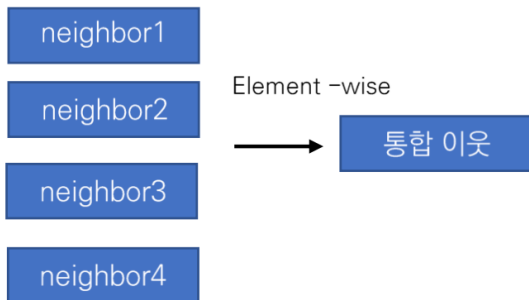
- Advantage of larger expressive capability
- Sequential architecture (not symmetric)
- Assigning an order to our unordered nodes by random permutation
- Being the slowest training time



# Aggregator function (3)

## □ Pooling aggregator

- Symmetric and trainable
- Applying element-wise max pooling
- Focus on single-layer architectures
- Effectively captures different aspects of the neighborhood set



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

# Learning the parameters of GraphSAGE

- Leveraging node feature information to efficiently generate node embeddings
- Graph-based Loss Function (unsupervised learning)
  - Encouraging nearby nodes to have similar representations
  - Enforcing the representations of disparate nodes to be highly distinct

$$J_G(\mathbf{z}_u) = -\log(\sigma(\mathbf{z}_u^\top \mathbf{z}_v)) - \frac{Q}{\mathbb{E}_{v_n \sim P_n(v)}} \log(\sigma(-\mathbf{z}_u^\top \mathbf{z}_{v_n}))$$

Generated output representations   No. of negative samples  Negative node of  $Z_u$

- Cross-Entropy Method for supervised learning :  $\mathcal{L}_{CE}(\mathbf{y}, \mathbf{p}) = -\sum_{i=1}^C y_i \log p_i.$

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.

Name	Citation		Reddit		PPI	
	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	<b>0.908</b>	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	<b>0.907</b>	<b>0.954</b>	0.482	<b>0.612</b>
GraphSAGE-pool	<b>0.798</b>	<b>0.839</b>	0.892	0.948	<b>0.502</b>	0.600
% gain over feat.	39%	46%	55%	63%	19%	45%

- GraphSAGE outperforms all baseline by a significant margin.
- Performance of unsupervised version is reasonably competitive with the fully supervised version.

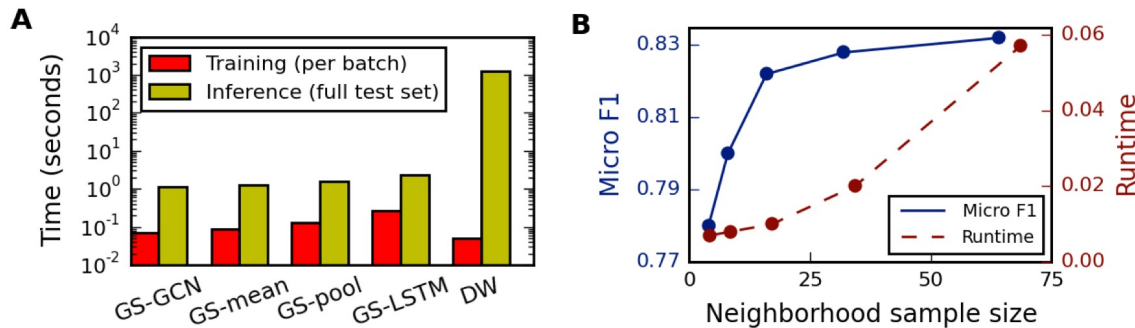


Figure 2: **A:** Timing experiments on Reddit data, with training batches of size 512 and inference on the full test set (79,534 nodes). **B:** Model performance with respect to the size of the sampled neighborhood, where the “neighborhood sample size” refers to the number of neighbors sampled at each depth for  $K = 2$  with  $S_1 = S_2$  (on the citation data using GraphSAGE-mean).

- (A) The training time for the methods are comparable with GraphSAGE being the slowest.
- (B) For sampling large neighborhoods, GraphSAGE is still able to maintain strong predictive accuracy, while improving the runtime.

# Conclusion

- Problem : Inability of existing node embedding approaches to generalize to unseen nodes in evolving or entirely new graphs, as they are inherently transductive.
- GraphSAGE is a general inductive framework that learns aggregator functions to generate node embeddings.
  - By sampling and aggregating features from a node's local neighborhood.
- The effectiveness of this approach is demonstrated through its superior performance on three inductive node-classification benchmarks.

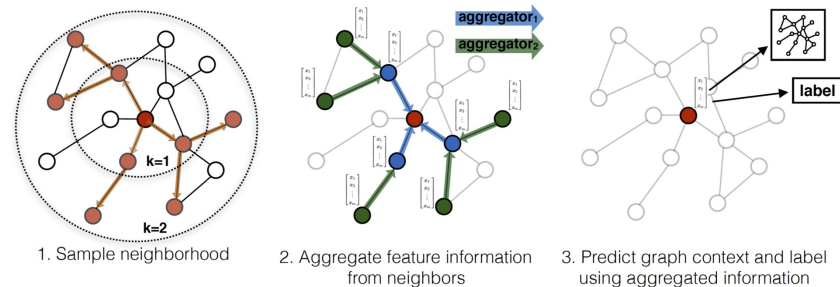


Figure 1: Visual illustration of the GraphSAGE sample and aggregate approach.

# Reference



- Hamilton, W., Ying, R. and Leskovec, J. (2018). “[Inductive Representation Learning on Large Graphs.](#)”
- Mishra et. al. “Node Masking: Making Graph Neural Networks Generalize and Scale Better”
- CS224W: GraphSAGE Neighbor Sampling
- DSBA, Paper Review MultiSAGE
- Paper Review from Gorio Learning



*Thank You!*



HTET ARKAR (hak3601@cau.ac.kr)