GraphSAGE

Inductive Representation Learning on Large Graphs

By William L. Hamilton, Rex Ying, Jure Leskovec

Hoyoung Choi 2022.02.03

INDEX

Introduction

Low-dimensional vector embedding is useful for graph analysis tasks

Limitations of previous works

1. Computational cost
   - GCN: For large graphs, excessive memory usage and computational time
2. Application to inductive learning
   - Previous works directly optimize the embeddings for each node
   - MF based models require additional computations
   - GCN has only been applied to transductive settings

Introduction

Inductive vs. Transductive Learning

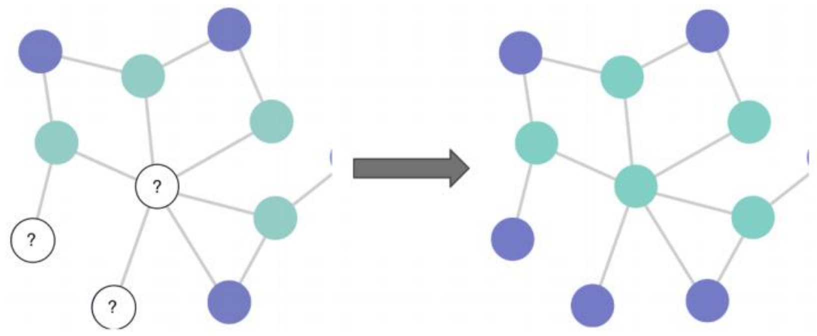
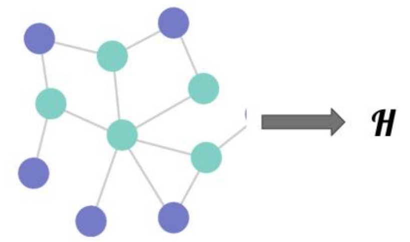
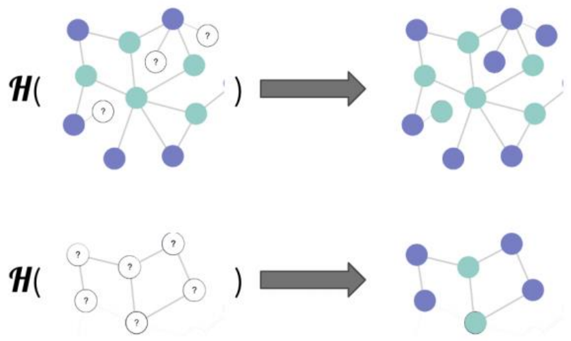


Figure 1. Node classification in transductive setting. At training time, the learning algorithm has access to all the nodes and edges including nodes for which labels are to be predicted.

Transductive



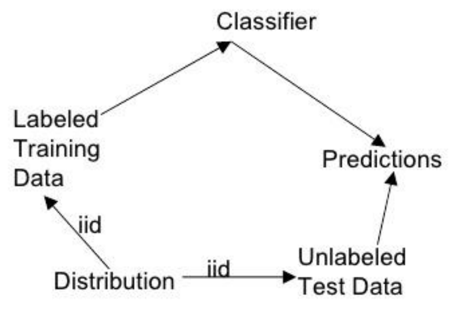
(a) A model $\mathcal{H}$ is learned over some graph



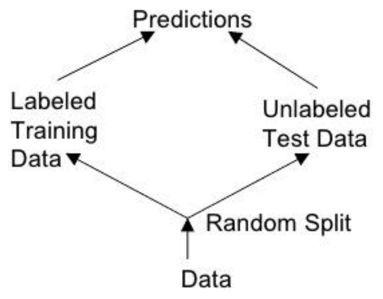
(b) The model is then by applied to new nodes and edges

Figure 2. Node classification in inductive setting. Once learned, the model can be applied to new unseen nodes (outlined in red). There may or may not exist edges between such new nodes and the nodes used for training.

Inductive



Inductive Learning



Transductive Learning

Introduction

GraphSAGE: SAmple and aggreGatE

- Learn a generalized embedding function
- Both learn distribution and topological structure
- Aggregator functions
- New loss function for unsupervised learning
- Both applied to supervised and unsupervised learning
- Perform better for evolving graphs

Related Work

Factorization-based embedding approaches

- Directly train node embeddings, need for excessive training in transductive settings
- Embedding space does not generalize

Graph convolutional networks

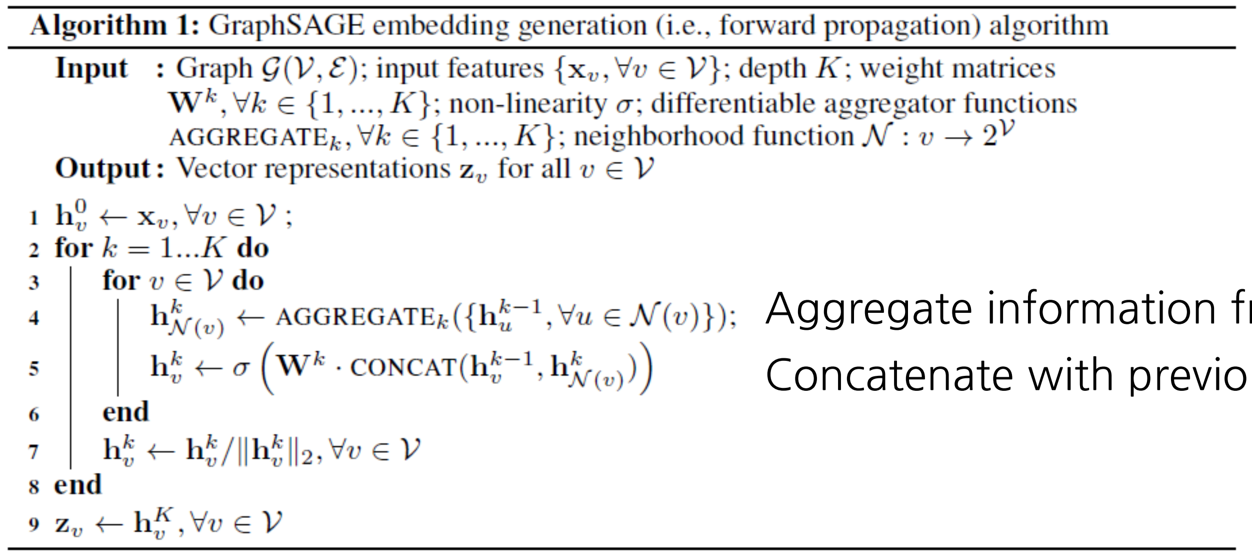
- Designed for semi-supervised learning in a transductive setting
- Need for full graph Laplacian
- GraphSAGE extended GCN to inductive settings

GraphSAGE

Embedding generation algorithm

Model already trained, parameters fixed

$\mathrm{AGGREGATE}_K$: K aggregator functions, W^K: Weight matrices



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 $\mathrm{AGGREGATE}_k, \forall k \in \{1,...,K\}$; neighborhood function $\mathcal{N} : v \to 2^{\mathcal{V}}$

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 $k = 1...K$ do
3     for $v \in \mathcal{V}$ do
4         $\mathbf{h}_{\mathcal{N}(v)}^k \leftarrow \mathrm{AGGREGATE}_k(\{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\})$;
5         $\mathbf{h}_v^k \leftarrow \sigma\left(\mathbf{W}^k \cdot \mathrm{CONCAT}(\mathbf{h}_v^{k-1}, \mathbf{h}_{\mathcal{N}(v)}^k)\right)$
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 $\mathbf{z}_v \leftarrow \mathbf{h}_v^K, \forall v \in \mathcal{V}$
```

Aggregate information from neighborhood

Concatenate with previous feature vector

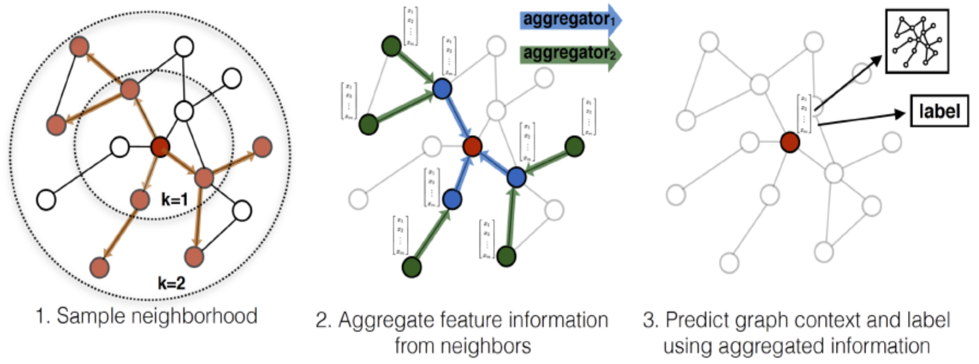
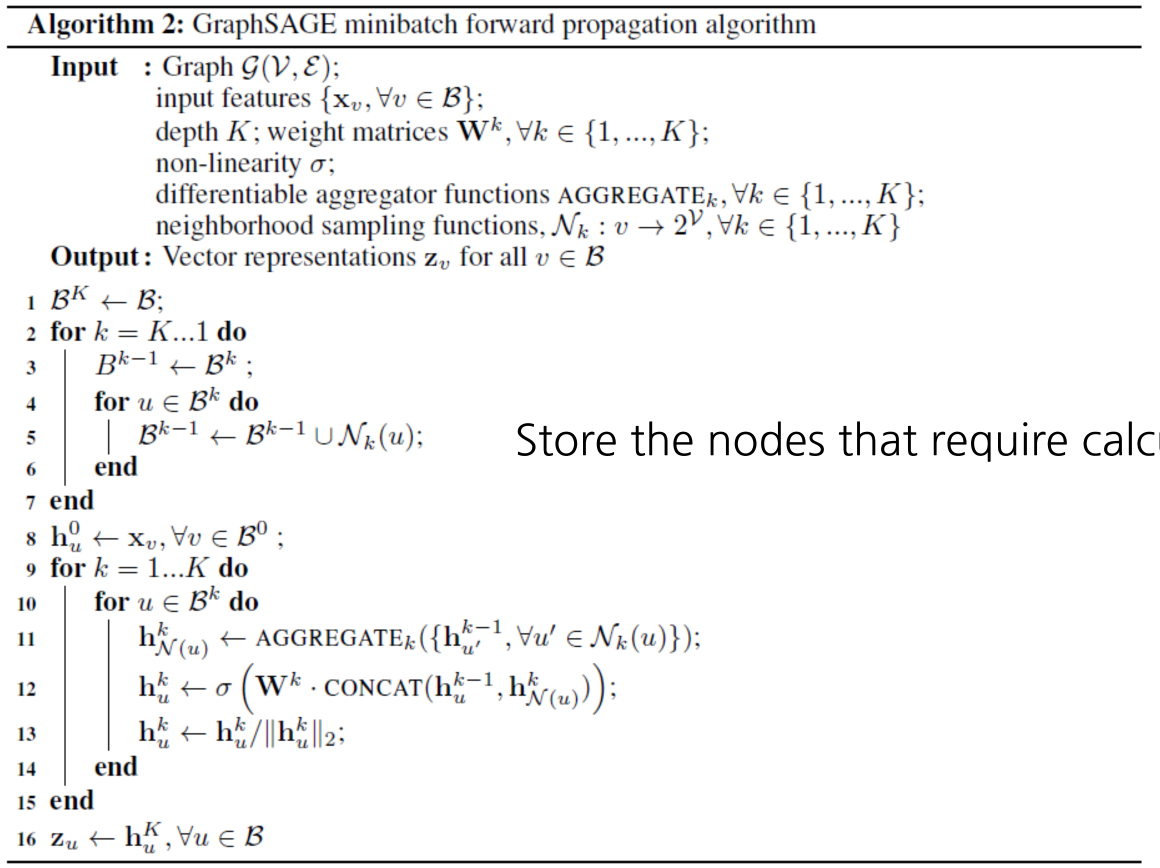

aggregator1
aggregator2
label
k=1
k=2
1. Sample neighborhood
2. Aggregate feature information from neighbors
3. Predict graph context and label using aggregated information

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

GraphSAGE

Embedding generation algorithm

Use mini batches to reduce the number of calculations



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 $\sigma$;
         differentiable aggregator functions $\text{AGGREGATE}_k, \forall k \in \{1, ..., K\}$;
         neighborhood sampling functions, $\mathcal{N}_k : v \to 2^{\mathcal{V}}, \forall k \in \{1, ..., K\}$
Output : Vector representations $\mathbf{z}_v$ for all $v \in \mathcal{B}$
1  $\mathcal{B}^K \leftarrow \mathcal{B}$;
2  for $k = K...1$ do
3      $\mathcal{B}^{k-1} \leftarrow \mathcal{B}^k$;
4      for $u \in \mathcal{B}^k$ do
5          $\mathcal{B}^{k-1} \leftarrow \mathcal{B}^{k-1} \cup \mathcal{N}_k(u)$;
6      end
7  end
8  $\mathbf{h}_u^0 \leftarrow \mathbf{x}_v, \forall v \in \mathcal{B}^0$;
9  for $k = 1...K$ do
10     for $u \in \mathcal{B}^k$ do
11         $\mathbf{h}_{\mathcal{N}(u)}^k \leftarrow \text{AGGREGATE}_k(\{\mathbf{h}_{u'}^{k-1}, \forall u' \in \mathcal{N}_k(u)\})$;
12         $\mathbf{h}_u^k \leftarrow \sigma\left(\mathbf{W}^k \cdot \text{CONCAT}(\mathbf{h}_u^{k-1}, \mathbf{h}_{\mathcal{N}(u)}^k)\right)$;
13         $\mathbf{h}_u^k \leftarrow \mathbf{h}_u^k / \|\mathbf{h}_u^k\|_2$;
14     end
15 end
16 $\mathbf{z}_u \leftarrow \mathbf{h}_u^K, \forall u \in \mathcal{B}$
```

Store the nodes that require calculations

Identical to previous pseudocode

GraphSAGE

Embedding generation algorithm

Neighborhood Definition

- Use a fixed-size set of neighbors that are uniformly selected
- $O\left(\prod_{i=1}^{K} S_i\right), S_i$ is the size of the neighborhood set for depth K
- Work best for $K = 2, S_1 \times S_2 \leq 500$ (Trade-off between efficiency and accuracy)

Why not using full neighborhood?

- Memory and expected running time is unpredictable
- Worst Case: $O(|\mathcal{V}|)$

GraphSAGE

Learning parameters of GraphSAGE

Loss function for unsupervised learning: graph-based loss

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

v: node that co-occur near u on fixed-length random walk

P_n: negative sampling distribution

v_n: negative samples

Q: number of negative samples

- Make the linked nodes lie in similar position, and non-linked nodes in different positions
- Train aggregator functions and weight matrices with SGD

GraphSAGE

Aggregator architectures

Nodes and its neighbors have no ordering

-> Aggregator should be symmetric (invariant to permutations of its inputs)

Symmetric, Trainable, With high representational capacity

- Mean Aggregator
- LSTM Aggregator
- Pooling Aggregator

GraphSAGE

Aggregator architectures

Mean Aggregator

- Take the elementwise mean of neighborhood nodes
- Similar to GCN, but Concatenating instead of Adding

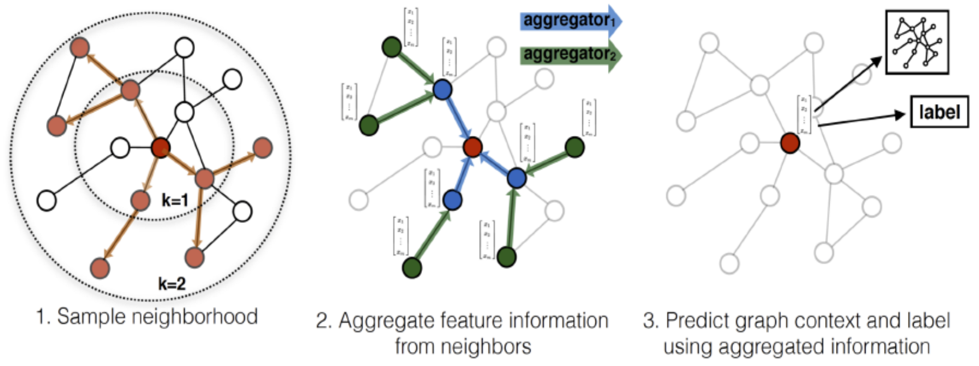

aggregator₁
aggregator₂
k=1
k=2
label
1. Sample neighborhood
2. Aggregate feature information from neighbors
3. Predict graph context and label using aggregated information

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

- Inductive version of GCN: Adding instead of Concatenating

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

GraphSAGE

Aggregator architectures

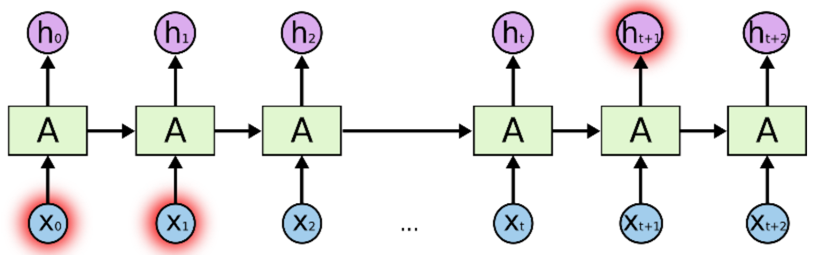
LSTM(Long Short-Term Memory) Aggregator

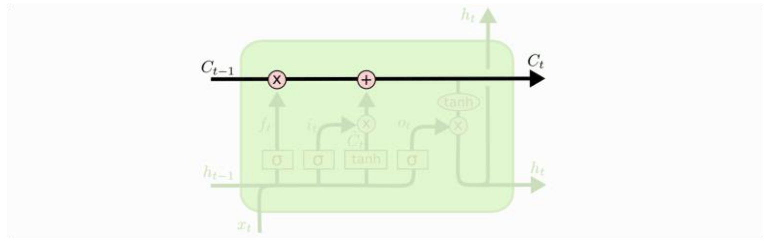
LSTM: variant of RNN (Recursive Neural Net)

- Long-term dependencies of RNN
- Use cell-state to pass information of past nodes

LSTM is sequential

- Use random permutations of neighbors





GraphSAGE

Aggregator architectures

Pooling Aggregator

- Symmetric and Trainable
- Fed neighbor's vector through fully connected neural net
- Element-wise max operator

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

- Max operator can be replaced with any symmetric vector function (ex) mean)
  - Max operator usually performs better

Experiments

Models and Loss function

Models

- Random classifier
- Logistic regression feature-based classifier
- DeepWalk (representative factorization-based approach)
- Concatenation of the raw features with DeepWalk
- GraphSAGE: GCN, mean, LSTM, pooling ($K = 2, S_1 = 25, S_2 = 10$)

Loss function

- Supervised setting: Cross-entropy loss
- Unsupervised setting: Graph-based loss

Experiments

Dataset

Citation data

- Predict paper subject categories on a large citation dataset

Reddit data

- Predict which community Reddit posts belong to
- Posts connected if the same user comments on both

Protein-Protein Interaction Data

- Learn node roles rather than community structure
- Classify protein roles

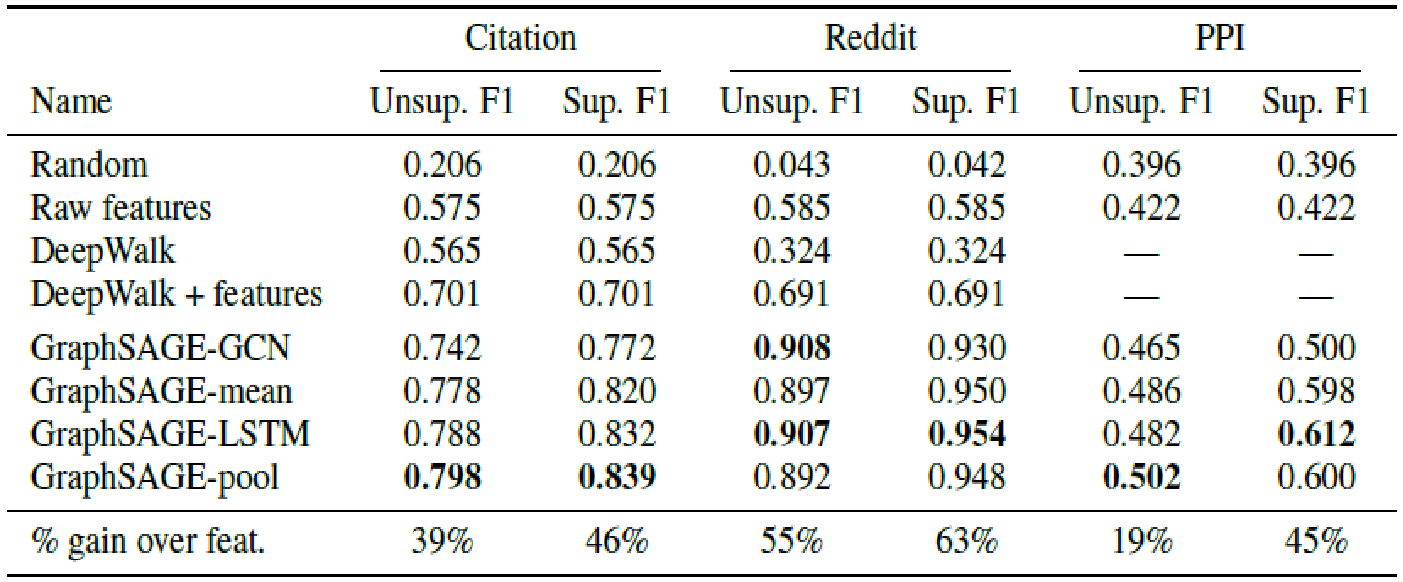
Experiments

Results

GraphSAGE performed better than baselines

LSTM and pooling performed better than GCN or mean

LSTM, pooling > mean > GCN



	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%

Experiments

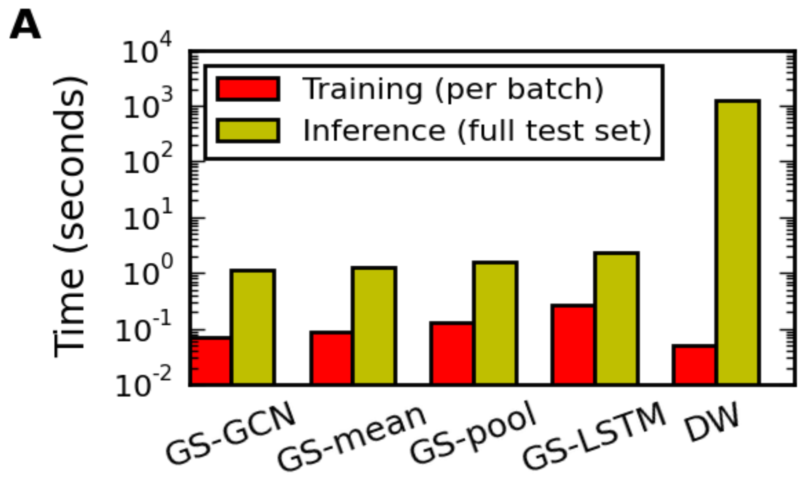
Runtime and parameter sensitivity

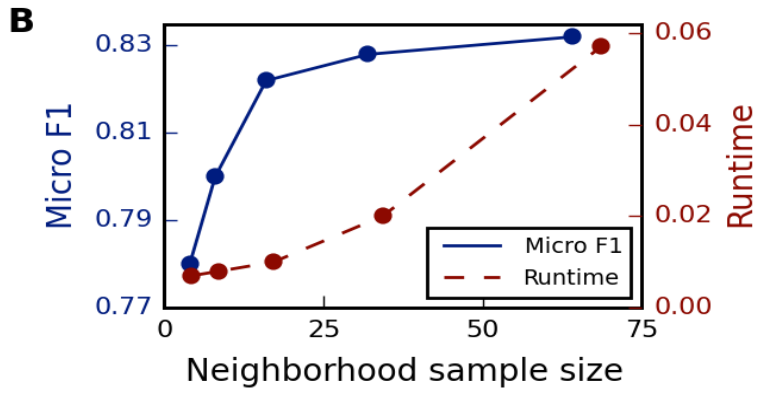
GraphSAGE is faster in inductive settings

For depth K

- $K = 2$ performed better than $K = 1$ (10~15%)
- Increase beyond 2 is meaningless

Sampling size and accuracy





Conclusion

GraphSAGE

- An efficient algorithm for generating embeddings from unseen nodes
- Effectively trades off performance and runtime
- Number of potential improvements are possible

Thank you