Qualifying Oral Exam: Representation Learning on Graphs

Pengyu Cheng

Duke University

April 5, 2020

Overview

Representation learning is an important task in machine learning.

Learning embeddings for images, videos, and other data with regular grid shapes has been well-studied.

There are tremendous real-world data with non-regular shapes, *e.g.* social networks, 3D point clouds, and knowledge graphs.

Graph is an effective mathematical tool to describe non-regular data.

Three reviewed papers are fundamental work in deep graph representation learning.

Overview

1. Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering [Defferrard et al., 2016]
   - Convolutional Networks on Graphs
   - Pooling on Graph Signal
   - Numerical Experiments
   - Discussion and Future Work
2. Semi-supervised Classification with Graph Convolutional Networks [Kipf and Welling, 2016]
   - Introduction
   - Approximation of convolutions on Graphs
   - Experiments
   - Discussion and Future Work
3. Inductive Representation learning on Large Graphs [Hamilton et al., 2017]
   - Introduction
   - Proposed Method: GraphSAGE
   - Experiments

Problem Description

Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering [Defferrard et al., 2016]

- Convolutional neural network(CNN) is an important technique to learn meaningful local patterns.
- CNNs are widely used on images, voices, videos, and other data with regular grid shapes.
- However, CNNs are inapplicable to non-Euclidean data.
- This paper give a solution of generalizing the convolution and the pooling operations of CNNs on graphs.

Preliminary

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E}, W)$ be a undirected graph with node set $\mathcal{V}$, $n = |\mathcal{V}|$, and edge set $\mathcal{E}$.

$W \in \mathbb{R}^{n \times n}$ is a weighted adjacency matrix.

The graph Laplacian is $\boldsymbol{L} = \boldsymbol{D} - \boldsymbol{W}$,
with normalized version as $\boldsymbol{L} = \boldsymbol{I}_n - \boldsymbol{D}^{-1/2}\boldsymbol{W}\boldsymbol{D}^{-1/2}$,
where $\boldsymbol{D}$ is diagonal degree matrix, and $\boldsymbol{I}_n$ is the identity matrix.

$\boldsymbol{L}$ is symmetric positive semi-definite, $\boldsymbol{L} = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^T$,
where $\boldsymbol{\Lambda} = \text{diag}([\lambda_0, \ldots, \lambda_{n-1}])$ are eigenvalues and
$\boldsymbol{U} = [u_0, \ldots, u_{n-1}]$ are eigenvectors.

Preliminary

Suppose $\boldsymbol{x} = [x_1, \dots x_n]^T \in \mathbb{R}^n$ is a graph signal,
where x_i is corresponding to node $v_i \in \mathcal{V}$.

The graph Fourier transformation for $\boldsymbol{x}$ is $\hat{\boldsymbol{x}} = \boldsymbol{U}^T\boldsymbol{x}$.
By $\boldsymbol{U}\boldsymbol{U}^T = \boldsymbol{I}_n$, the inverse graph Fourier transformation $\boldsymbol{x} = \boldsymbol{U}\hat{\boldsymbol{x}}$.

For classic Fourier transform, convolution in signal domain equals to point-wise multiplication in spectral domain then transforming back.

Definition of graph convolution $*_{\mathcal{G}}$,

$$\boldsymbol{x} *_{\mathcal{G}} \boldsymbol{y} = \boldsymbol{U}((\boldsymbol{U}^T\boldsymbol{x}) \odot (\boldsymbol{U}^T\boldsymbol{y})) = \boldsymbol{U}[\text{diag}(\boldsymbol{U}^T\boldsymbol{x})]\boldsymbol{U}^T\boldsymbol{y}, \qquad (1)$$

with $\odot$ being point-wise multiplication.

Non-parametric Convolution Filters

Considering a graph convolutional filter in **spectral** domain, with parameters $\boldsymbol{\theta} \in \mathbb{R}^n$ as $g_{\boldsymbol{\theta}}(\boldsymbol{L}) = \text{diag}(\boldsymbol{\theta})$.

Then the convolution between signal $\boldsymbol{x}$ with filter $g_{\boldsymbol{\theta}}$ is written as

$$g_{\boldsymbol{\theta}} *_{\mathcal{G}} \boldsymbol{x} = \boldsymbol{U} g_{\boldsymbol{\theta}}(\boldsymbol{L}) \boldsymbol{U}^T \boldsymbol{x} = \boldsymbol{U} \text{diag}(\boldsymbol{\theta}) \boldsymbol{U}^T \boldsymbol{x}. \tag{2}$$

This non-parametric filter has two disadvantages:
(1) could not ensure extracting information in a local
(*i.e.* information from a node with its close neighbors);
(2) its parameter size is $\mathcal{O}(n)$, increasing with the node number.

Polynomial Parametrization

To solve those problems, the authors propose parameterized filters:

$$g_{\boldsymbol{\theta}}(\boldsymbol{L}) = \sum_{k=0}^{K-1} \theta_k \boldsymbol{\Lambda}^k, \tag{3}$$

The convolution for a signal $\boldsymbol{x}$ is

$$g_{\boldsymbol{\theta}} *_{\mathcal{G}} \boldsymbol{x} = \boldsymbol{U}[\sum_{k=0}^{K-1} \theta_k \boldsymbol{\Lambda}^k]\boldsymbol{U}^T\boldsymbol{x} = \sum_{k=0}^{K-1} \theta_k \boldsymbol{L}^k \boldsymbol{x}. \tag{4}$$

Hammond et al. [2011] show that if $d_{\mathcal{G}}(i,j) > K$, then $[\boldsymbol{L}^K]_{ij} = 0$, where $d_{\mathcal{G}}$ is the length of the shortest path from v_i to v_j on graph $\mathcal{G}$.

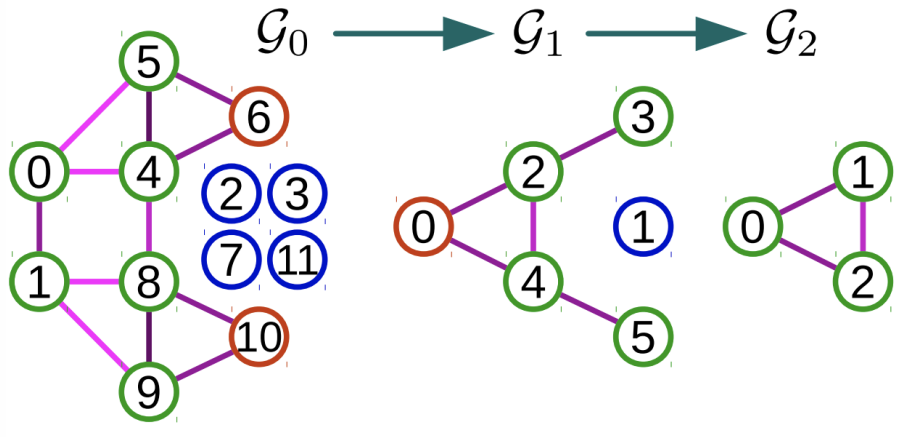
Therefore, each node is only involved with neighbors whose distance to it is smaller than K.
Also, learning complexity of $g_{\boldsymbol{\theta}}$ becomes $\mathcal{O}(K)$, as a constant to the node size n.

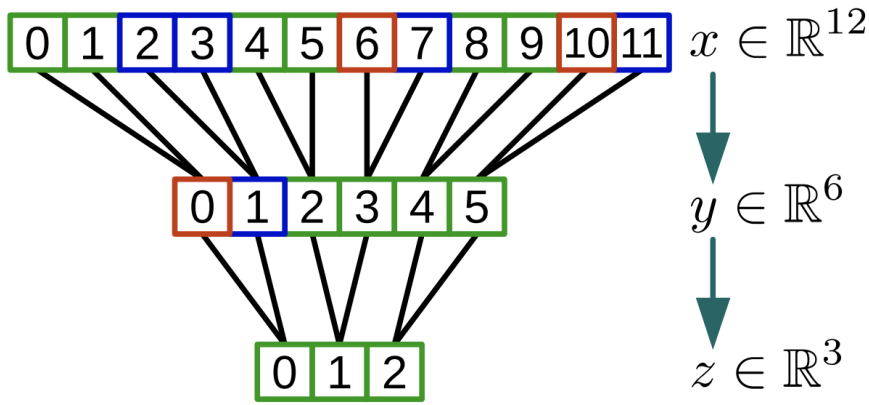
Pooling on Graph Signal

Based on idea: similar vertices are supposed to be clustered together.

Graclus multi-level clustering: at each level $\mathcal{G}_h$:
(1) randomly selects a unmarked node;
(2) matches the node to a unmarked neighbor maximizing normalized edge cut $W_{ij}(1/d_i + 1/d_j)$;
(3) marks the matched two nodes.

The operation is repeated until all nodes becomes marked.



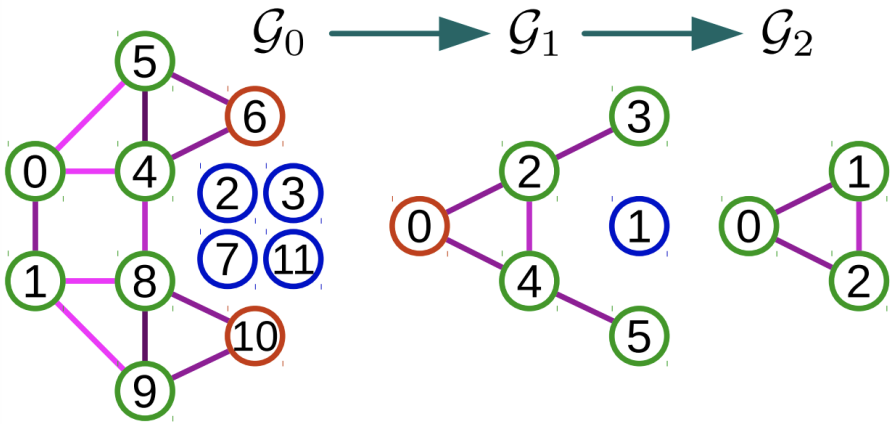


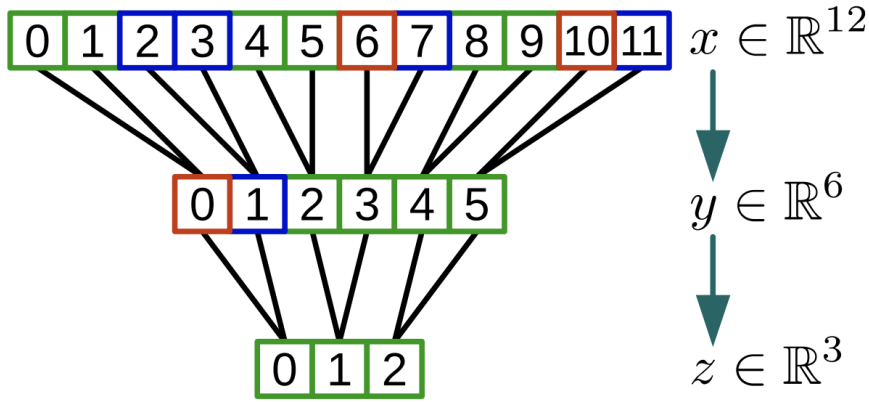
Pooling on Graph Signal

Pooling operation:
frequently applied during training $\rightarrow$ large computational complexity.

Efficient solution: record the pooling assignments before training.

Build a binary tree to record node matching assignments:
(1) If $v_i^{(h)}, v_j^{(h)} \in \mathcal{G}_h$ are pooled to $v_l^{(h+1)} \in \mathcal{G}_{h+1}$,
store $v_i^{(h)}, v_j^{(h)}$ as children of $v_l^{(h+1)}$ on binary tree.
(2) Assign fake nodes to not matched nodes.





Experiments

Comparison with original CNNs on MNIST.

Converting images to graph:
represent each pixel by a node; connect to 8 nearest neighbors.

The weighted adjacency matrix $\boldsymbol{W}$ is defined as

$$[\boldsymbol{W}]_{ij} = \exp(-\frac{\|z_i - z_j\|_2^2}{\sigma^2}), \tag{5}$$

where z_i is the pixel value of the i-th pixel.

Model	Architecture	Accuracy
Classical CNN	C32-P4-C64-P4-FC512	99.33
Proposed graph CNN	GC32-P4-GC64-P4-FC512	99.14

Architecture	8-NN on 2D Euclidean grid	random
GC32	97.40	96.88
GC32-P4-GC64-P4-FC512	99.14	95.39

Experiments

Besides, the graph CNNs have rotational invariance which CNNs for regular grids do not have.

Apply to Text classification:
The 20News dataset contains
18846 documents with 20 class labels.
represent each document $\boldsymbol{x}$ as a graph.

Model	Accuracy
Linear SVM	65.90
Multinomial Naive Bayes	68.51
Softmax	66.28
FC2500	64.64
FC2500-FC500	65.76
GC32	68.26

Each word is a node and nodes are
connected to 16 nearest neighbor based
on the similarity of their Word2Vec embeddings.

	word2vec			
bag-of-words	pre-learned	learned	approximate	random
67.50	66.98	68.26	67.86	67.75

Discussion and Future Work

Some directions to improve the proposed model.

- The pooling requires a weighted adjacency matrix as a measurement to pair nodes, $W_{ij}(1/d_i + 1/d_j)$.
  However, a large number of graphs do not have this additional information.
- To record the pooling assignment, the model builds a binary tree. When new graphs come or the structures of graphs change, the model need to rebuild the binary tree, which leads to high computational complexity.

Therefore, how to efficiently operate pooling on graphs still remains an interesting problem.

Introduction

Semi-supervised Classification with Graph Convolutional Networks [Kipf and Welling, 2016]

In this paper, the authors simplify the graph convolution with a first-order approximation called Graph Convolutional Network(GCN).

Then the new method shows effective experiment results on semi-supervised node classification tasks.

Recall the convolution filter $g_{\boldsymbol{\theta}}(\boldsymbol{L}) = \sum_{k=0}^{K-1} \theta_k \boldsymbol{\Lambda}^k$.
The Convolution layer

$$g_{\boldsymbol{\theta}} *_{\mathcal{G}} \boldsymbol{x} = \boldsymbol{U}[\sum_{k=0}^{K-1} \theta_k \boldsymbol{\Lambda}^k]\boldsymbol{U}^T \boldsymbol{x} = \sum_{k=0}^{K-1} \theta_k \boldsymbol{L}^k \boldsymbol{x}.$$

Simplify the convolution with the polynomial order $K = 1$.

Convolution Approximation

Three explanations to approximation:

- stacking multiple convolutional layers with $K = 1$ can reach the similar performance as high-order convolutions
- low-order convolution reduce the over-fitting when applying to graphs with large-range node degree distributions
- with a limited computational ability, the $K = 1$ approximation allows deeper models, improving modeling capacity.

replace weighted adjacency matrix $\boldsymbol{W}$ with adjacency matrix $\boldsymbol{A}$,

$$g_{\boldsymbol{\theta}} *_{\mathcal{G}} \boldsymbol{x} = \theta_1 \boldsymbol{L}\boldsymbol{x} + \theta_0 \boldsymbol{x} = \theta'_0 \boldsymbol{x} - \theta'_1 \boldsymbol{D}^{-1/2}\boldsymbol{A}\boldsymbol{D}^{-1/2}\boldsymbol{x}, \tag{6}$$

The second approximation: let $\theta = \theta'_0 = -\theta'_1$,

$$g_{\boldsymbol{\theta}} *_{\mathcal{G}} \boldsymbol{x} = \theta \left(\boldsymbol{I}_n + \boldsymbol{D}^{-1/2}\boldsymbol{A}\boldsymbol{D}^{-1/2}\right) \boldsymbol{x}. \tag{7}$$

Convolution Approximation

To increase the numerical stability, the authors introduce the re-normalization trick $\boldsymbol{I}_n + \boldsymbol{D}^{-1/2}\boldsymbol{A}\boldsymbol{D}^{-1/2} \rightarrow \tilde{\boldsymbol{D}}^{-1/2}\tilde{\boldsymbol{A}}\tilde{\boldsymbol{D}}^{-1/2}$, where $\tilde{\boldsymbol{A}} = \boldsymbol{A} + \boldsymbol{I}_n$ and $\tilde{\boldsymbol{D}} = \boldsymbol{D} + \boldsymbol{I}_n$.

Convolution for signals with multiple channels: $\boldsymbol{X} \in \mathbb{R}^{n \times c}$ and outputs $\boldsymbol{Z} \in \mathbb{R}^{n \times f}$, generalize Eq. (7) with parameter $\boldsymbol{\Theta}$

$$\boldsymbol{Z} = \tilde{\boldsymbol{D}}^{-1/2}\tilde{\boldsymbol{A}}\tilde{\boldsymbol{D}}^{-1/2}\boldsymbol{X}\boldsymbol{\Theta} \tag{8}$$

The authors use Eq.(7) to solve the semi-supervised node classification problem. The model is a two-layer GCN:

$$\boldsymbol{Z} = f(\boldsymbol{X}, \boldsymbol{A}) = \text{softmax}\left(\hat{\boldsymbol{A}}\ \text{ReLU}\left(\hat{\boldsymbol{A}}\boldsymbol{X}\boldsymbol{W}^{(0)}\right)\boldsymbol{W}^{(1)}\right), \tag{9}$$

where $\hat{\boldsymbol{A}} = \tilde{\boldsymbol{D}}^{-1/2}\tilde{\boldsymbol{A}}\tilde{\boldsymbol{D}}^{-1/2}$.
The loss function $\mathcal{L} = -\sum_{l \in \mathcal{Y}_L}\sum_{f=1}^{F} Y_{lf} \log Z_{lf}$ where $\mathcal{Y}_L$ is the labeled node set, and each Y_l is a one-hot node label for l-th node.

Experiments

The model is trained with full gradient descent. The authors conduct the semi-supervised node classification on citation networks and knowledge graphs:

Dataset	Type	Nodes	Edges	Classes	Features	Label rate
Citeseer	Citation network	3,327	4,732	6	3,703	0.036
Cora	Citation network	2,708	5,429	7	1,433	0.052
Pubmed	Citation network	19,717	44,338	3	500	0.003
NELL	Knowledge graph	65,755	266,144	210	5,414	0.001

Method	Citeseer	Cora	Pubmed	NELL
ManiReg [3]	60.1	59.5	70.7	21.8
SemiEmb [28]	59.6	59.0	71.1	26.7
LP [32]	45.3	68.0	63.0	26.5
DeepWalk [22]	43.2	67.2	65.3	58.1
ICA [18]	69.1	75.1	73.9	23.1
Planetoid* [29]	64.7 (26s)	75.7 (13s)	77.2 (25s)	61.9 (185s)
GCN (this paper)	**70.3** (7s)	**81.5** (4s)	**79.0** (38s)	**66.0** (48s)
GCN (rand. splits)	67.9 ± 0.5	80.1 ± 0.5	78.9 ± 0.7	58.4 ± 1.7

Instead of fixing the labeled node set, the authors also provide results that randomly select the labeled node set (rand.splits).

Experiments

Besides, the authors study the performance of different convolution approximations and report the mean classification accuracy on citation networks.
From the table 1, the original GCN (re-normalization trick) shows the best performance.

Description		**Propagation model**	**Citeseer**	**Cora**	**Pubmed**
Chebyshev filter (Eq. 5)	$K=3$	$\sum_{k=0}^{K} T_k(\tilde{L})X\Theta_k$	69.8	79.5	74.4
	$K=2$		69.6	81.2	73.8
1^{st}-order model (Eq. 6)		$X\Theta_0 + D^{-\frac{1}{2}}AD^{-\frac{1}{2}}X\Theta_1$	68.3	80.0	77.5
Single parameter (Eq. 7)		$(I_N + D^{-\frac{1}{2}}AD^{-\frac{1}{2}})X\Theta$	69.3	79.2	77.4
Renormalization trick (Eq. 8)		$\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}X\Theta$	**70.3**	**81.5**	**79.0**
1^{st}-order term only		$D^{-\frac{1}{2}}AD^{-\frac{1}{2}}X\Theta$	68.7	80.5	77.8
Multi-layer perceptron		$X\Theta$	46.5	55.1	71.4

Figure: Comparison of different propagation models

Discussion and Future Work

In this paper, the authors provide a simplified graph convolution with first-order approximation.

The authors compare their method with standard graph convolution and show the GCN method has a better numerical performance.

A important future work directions is to reduce the memory usage.

The GCN model requires full-batch gradient descent, which is high computationally complex, especially when the size of networks goes larger.

Designing a Mini-batch version GCN is a meaningful direction to study.

Introduction

Inductive Representation learning on Large Graphs [Hamilton et al., 2017].

The authors proposed a new inductive graph representation learning method called GraphSAGE.

When learning representation for one node, the GraphSAGE aggregates information from its neighbors.

This learning strategy enables the model to assign embeddings to unseen new nodes.

Besides, this method can scale to large graphs, where graph convolution networks are difficult to be applied.

Proposed Method: GraphSAGE

Assume K information aggregator functions $\{\text{AGGREGATE}_k\}_{k=1}^{K}$ have been well-trained.
The forward propagation of GraphSAGE is show in Algorithm 1.

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
$\mathbf{z}_v \leftarrow \mathbf{h}_v^K, \forall v \in \mathcal{V}$

Mini-batch scheme: Randomly select neighbors.

Loss Function

The complexity of neighbor selection in mini-batch setting becomes unpredictable. Randomly select a fix number of nodes from the node neighborhood, which is an unbiased estimation under expectation.

The authors use the output of GraphSAGE to predict the connection information from the original graph.

$$J_{\mathcal{G}}(z_u) = -\log(\sigma(z_u^T z_v)) - Q \cdot \mathbb{E}_{v_n \sim P_n(v)} \log(\sigma(-z_u^T z_{v_n})), \quad (10)$$

where v is one of the node u's neighbor, P_n is a negative sampling distribution, and Q represents the number of negative samples.

When other supervised information is provide, *e.g.* node labels for classification, this unsupervised loss can be replaced by other supervised learning losses.

Aggregator Architectures

The authors introduce three different aggregation functions:

Mean Aggregator: averaging the neighbors' embeddings:

$$\boldsymbol{h}_v^k \leftarrow \sigma(\boldsymbol{W} \cdot \text{MEAN}(\{\boldsymbol{h}_v^{k-1}\} \cup \{\boldsymbol{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\})). \tag{11}$$

LSTM Aggregator: using a LSTM to extract the neighbors' embeddings. The order of the neighbors to feed into LSTM is from a permutation of the neighbor node set.

Pooling Aggregator: feed neighbors' embeddings into a fully-connected layer, then do max-pooling:

$$\text{AGGREGATE}_k^{pool} = \max(\{\sigma(\boldsymbol{W}_{pool}\boldsymbol{h}_{u_i}^k + b), \forall u_i \in \mathcal{N}(v)\}). \tag{12}$$

Experiments

The proposed GraphSAGE is evaluated on three downstream tasks: (1) node classification on citation networks; (2) communities recognition for Reddit posts; (3) protein function classification on biological protein-protein interaction (PPI) graphs.
For citation networks and Reddit dataset, the authors test on the nodes that are unseen in the training process; for the PPI dataset, entire unseen graphs are tested.

	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%

Discussion and Future Work

In this paper, the authors introduce a inductive graph representation learning method, which is scalable for large graphs and effective to assign embeddings to unseen nodes.

However, some weakness still remains and is meaningful to study:

- For low degree nodes, the aggregation from neighbors causes a high variance of embeddings, especially for new-coming unseen nodes.
- Although the authors use fix neighborhood size, when the layer number K goes larger, the number of required neighbors will increase exponentially. This is challenging for mini-batch training. However, if reducing the neighbor selection size, the variance of embeddings will go larger. Therefore, how to solve the variance-complexity trade-off is worth to research.

Thank you!

Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in neural information processing systems*, pages 3844–3852, 2016.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, pages 1024–1034, 2017.

David K Hammond, Pierre Vandergheynst, and Rémi Gribonval. Wavelets on graphs via spectral graph theory. *Applied and Computational Harmonic Analysis*, 30(2):129–150, 2011.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.