Graph Neural Networks and Self-supervised Learning

Jie Tang

Computer Science

Tsinghua University

The slides can be downloaded at http://keg.cs.tsinghua.edu.cn/jietang

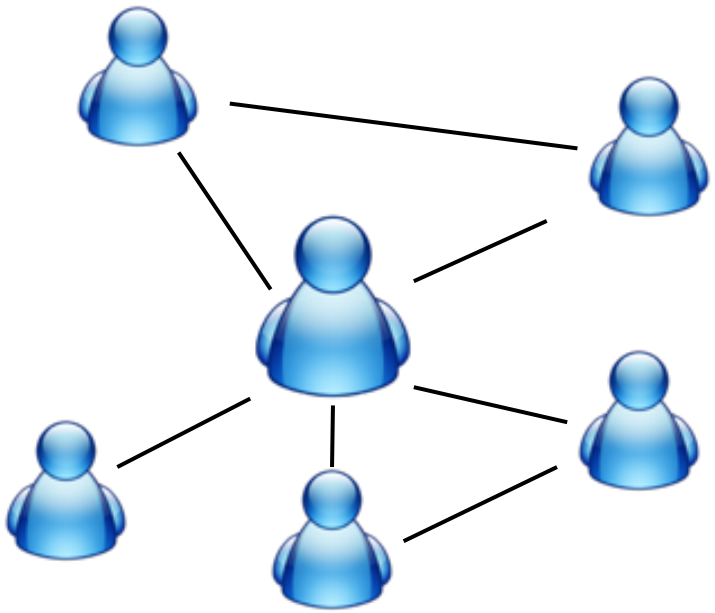
A bit about Jie

- Jie Tang, IEEE Fellow, Professor, Associate Chair of Dept. of CS at Tsinghua University. Interests include social network, data mining, machine learning, knowledge graph.
  - SIGKDD Test of Time Award
  - SIGKDD Service Award
  - NSFC for Distinguished Young Scholars, CCF Young Scientist Award
  - 2nd National S&T Award

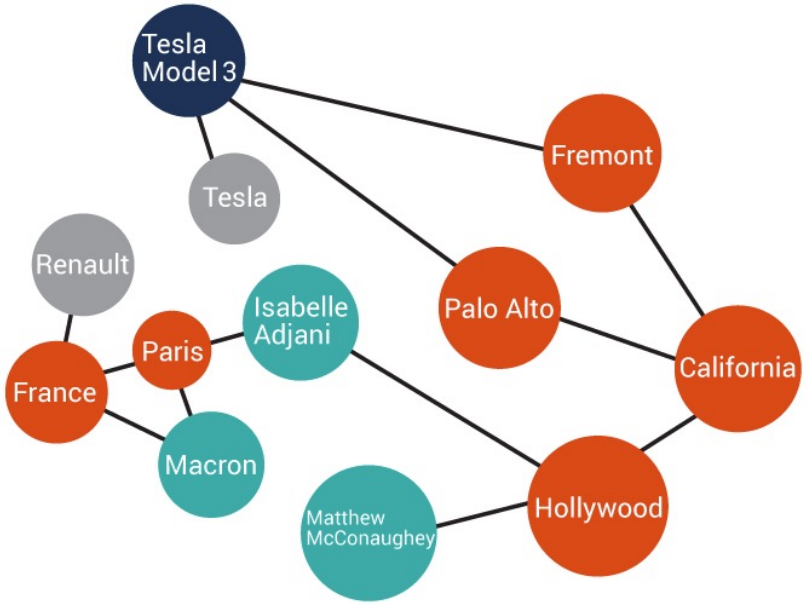
- PC Co-Chair
- Editor in Chie urnal
- Have publish DD (30), IJCAI/AAAI (30), NIPS/ICML (10+), IEEE Trans. (30), Machine Learning J
- #Citations: 20,000+ and *h*-index: 74
- ArnetMiner https://www.aminer.org/ for academic researcher network analysis, which attracted 20 million users from 220 countries/regions.
- **HP:** http://keg.cs.tsinghua.edu.cn/jietang/

Work Hard and Play Harder!

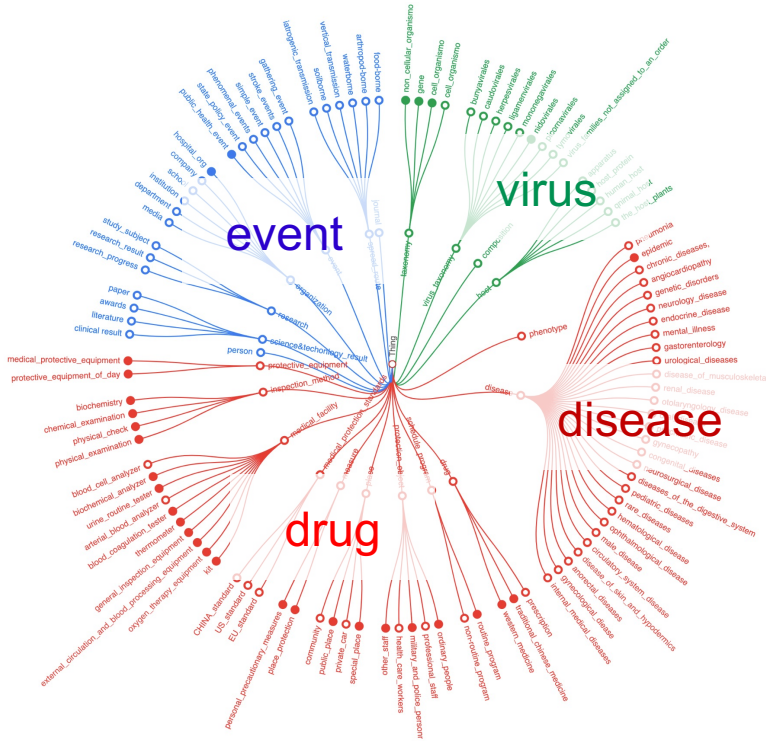
Networked Data



Social Network



Knowledge Graph



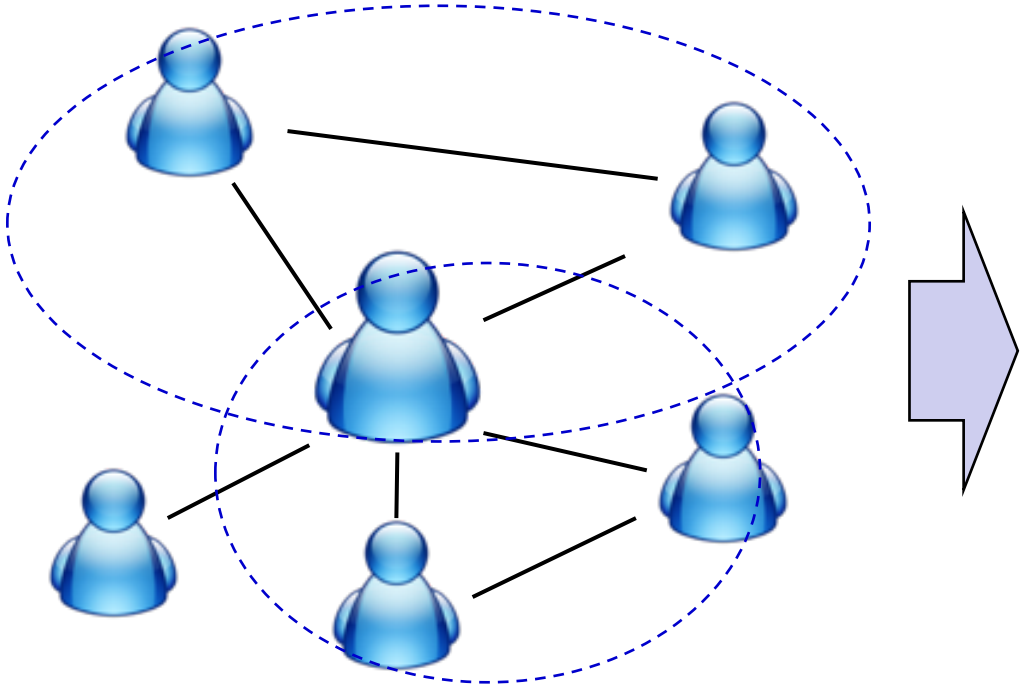
COVID Graph

Machine Learning with Networks

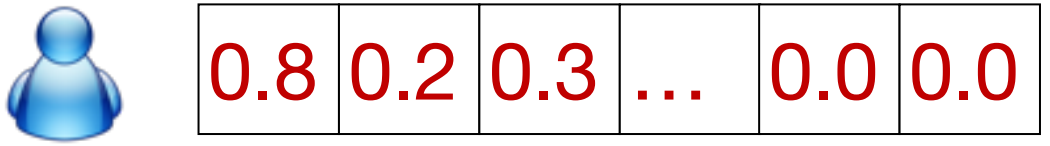
- ML tasks in networks:
  - Node classification
    - Predict a type of a given node
  - Link prediction
    - Predict whether two nodes are linked
  - Community detection
    - Identify densely linked clusters of nodes
  - Network similarity
    - How similar are two (sub)networks?

Representation Learning on Networks

Representation Learning/
Graph Embedding

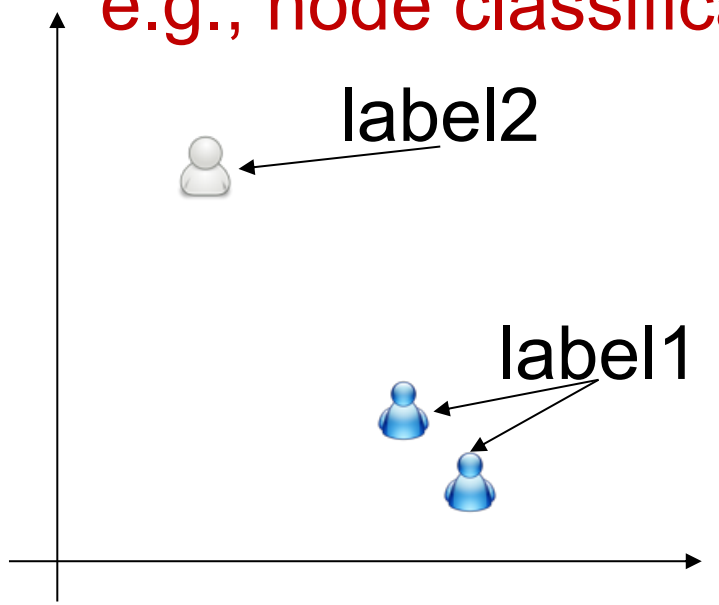


d-dimensional vector, $d << |V|$

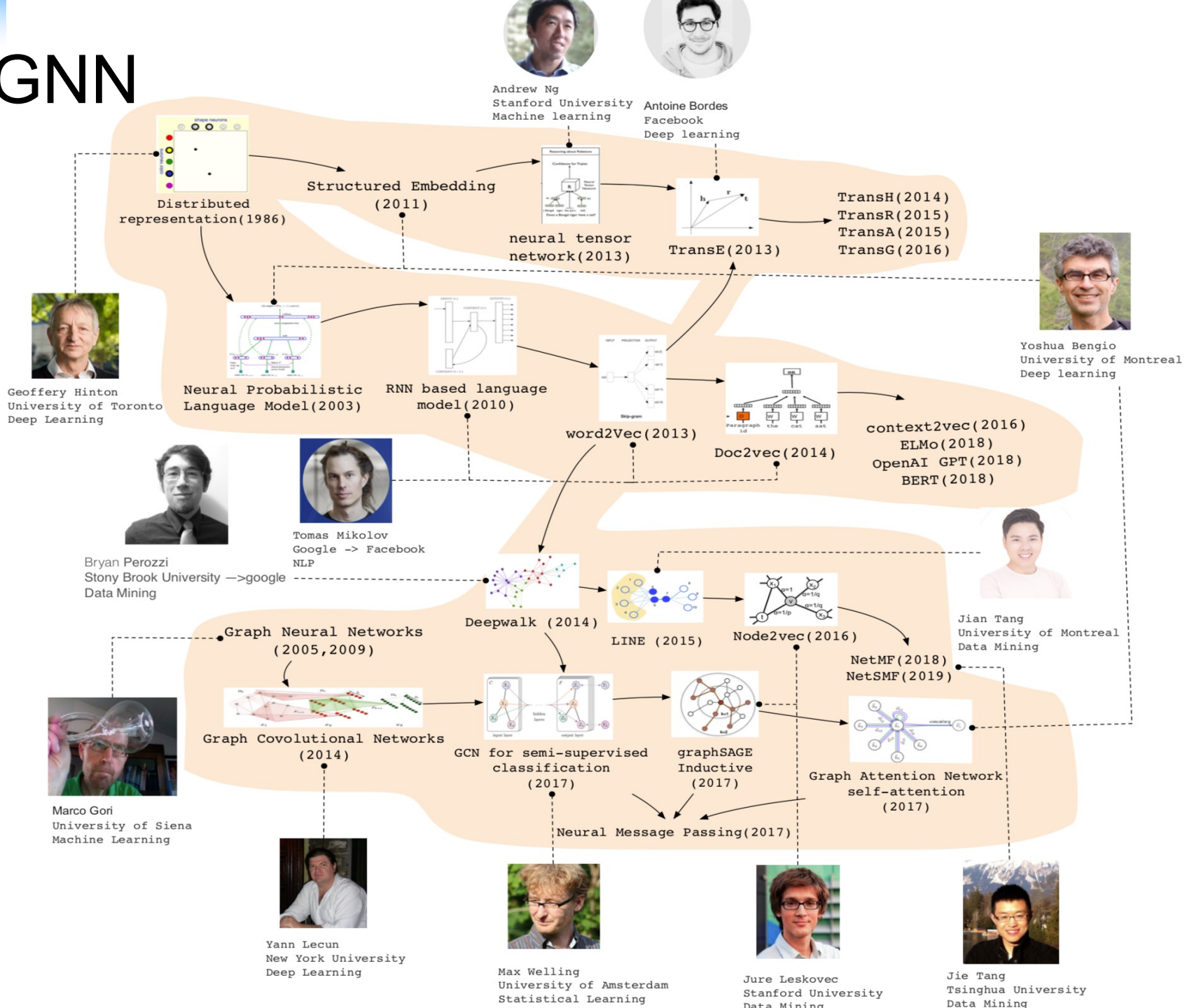


Users with the same label are located in closer

e.g., node classification



GRL: NE&GNN


Andrew Ng
Stanford University
Machine learning
Antoine Bordes
Facebook
Deep learning
Distributed representation(1986)
Structured Embedding (2011)
neural tensor network(2013)
TransE(2013)
TransH(2014)
TransR(2015)
TransA(2015)
TransG(2016)
Yoshua Bengio
University of Montreal
Deep learning
Geoffery Hinton
University of Toronto
Deep Learning
Neural Probabilistic Language Model(2003)
RNN based language model(2010)
word2Vec(2013)
Doc2vec(2014)
context2vec(2016)
ELMo(2018)
OpenAI GPT(2018)
BERT(2018)
Tomas Mikolov
Google -> Facebook
NLP
Bryan Perozzi
Stony Brook University —>google
Data Mining
Deepwalk (2014)
LINE (2015)
Node2vec(2016)
Jian Tang
University of Montreal
Data Mining
NetMF(2018)
NetSMF(2019)
Graph Neural Networks (2005,2009)
Graph Covolutional Networks (2014)
GCN for semi-supervised classification (2017)
graphSAGE Inductive (2017)
Graph Attention Network self-attention (2017)
Marco Gori
University of Siena
Machine Learning
Neural Message Passing(2017)
Yann Lecun
New York University
Deep Learning
Max Welling
University of Amsterdam
Statistical Learning
Jure Leskovec
Stanford University
Data Mining
Jie Tang
Tsinghua University
Data Mining

超大规模图神经网络

① 基于图神经网络的大规模图数据挖掘

现实场景中充斥着大量图结构数据，涵盖社交网络、推荐系统、学术网络等，传统深度学习难以处理大规模图数据中复杂的结构化信息。清华大学计算机系知识工程实验室团队深入研究图神经网络，在图表示学习、图预训练模型、图神经网络鲁棒性、异构图学习等领域取得突破，为大规模图数据挖掘提供了理论依据和技术支持。

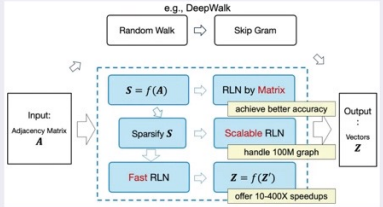
② 大规模图表示学习

如何从大规模图数据中抽取出有效的表示是图数据挖掘的一大难题。以往方法如 DeepWalk、LINE、Node2Vec 计算复杂度高并缺乏理论支持，团队从理论上证明此类方法实际上可以统一成矩阵分解的近似形式，并提出基于显式矩阵分解的 NetMF 方法，效果有显著提升。NetSMF 进一步将该方法扩展到稀疏图数据上，并在百万级节点规模的图数据实验中，将原本所需数月的计算缩短到一天以内。

团队进一步优化基于稀疏矩阵分解的图表示学习方法，提出引入频域图嵌入的 ProNE，在上亿级节点图上计算效率提高上百倍。

团队还将图表示学习拓展到异构图中，提出结合多模态异构图属性的 GATNE，在四亿商品网络上以极高精度预测节点间关系，性能较同质图方法显著提升。

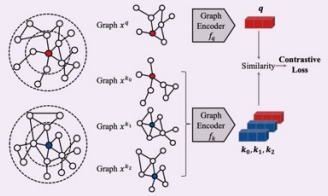
此外，为了更好地评价图神经网络在异构图上的性能，团队提出了异构图学习基准 HGB，大大推动了异构图神经网络的发展。



Algorithm	Closed Matrix Form
DeepWalk	$\log\left(\mathrm{vol}(G)\left(\frac{1}{T}\sum_{r=1}^{T}(D^{-1}A)^r\right)D^{-1}\right)-\log b$
LINE	$\log\left(\mathrm{vol}(G)D^{-1}AD^{-1}\right)-\log b$
PTE	$\log\left(\begin{pmatrix}\alpha\,\mathrm{vol}(G_{ww})(D_{row}^{ww})^{-1}A_{ww}(D_{col}^{ww})^{-1}\\ \beta\,\mathrm{vol}(G_{dw})(D_{row}^{dw})^{-1}A_{dw}(D_{col}^{dw})^{-1}\\ \gamma\,\mathrm{vol}(G_{lw})(D_{row}^{lw})^{-1}A_{lw}(D_{col}^{lw})^{-1}\end{pmatrix}\right)-\log b$
node2vec	$\log\left(\frac{\frac{1}{T}\sum_{r=1}^{T}\left(\sum_u X_{w,u}P_{r,w,u}^{r}+\sum_u X_{c,u}P_{w,c,u}^{r}\right)}{(\sum_u X_{w,u})(\sum_u X_{c,u})}\right)-\log b$

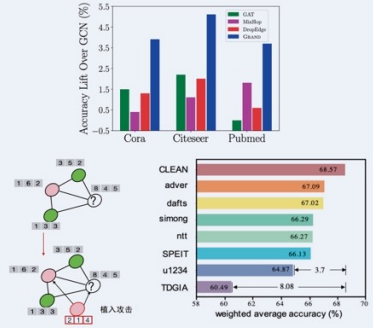
③ 大规模图预训练模型

图数据由于其独特的结构化属性，难以进行迁移。团队提出图预训练技术 GCC，将图结构信息通过随机采样加对比学习的方式进行编码，训练出适用于不同类型、不同规模图数据的预训练模型。该模型在大规模图数据上有效提高节点分类、子图分类等下游任务上的表现。



④ 图神经网络鲁棒性

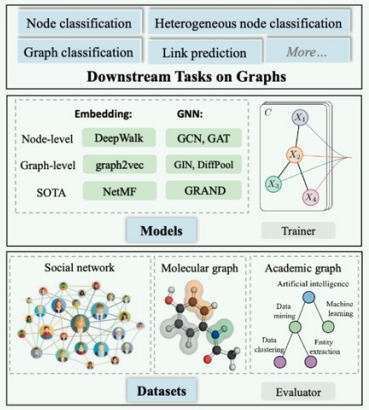
图神经网络容易受到对抗攻击风险，团队从攻防两方面展开鲁棒性研究。提出了随机图神经网络 GRAND，通过随机信息传播过程减轻攻击带来的影响；提出了拓扑图植入攻击 TDGIA，针对图数据中的薄弱连接，注入少量节点就能大幅降低图神经网络性能。



(c) TDGIA vs. The Best Results in KDD-CUP 2020

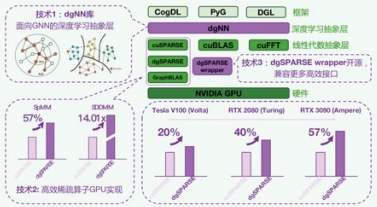
⑤ 高效易用的图深度学习框架CogDL

为了更好推进图神经网络发展，团队提出了 CogDL 图深度学习框架。CogDL 的核心理念包括易用接口、可复现性和高效计算，帮助研究人员可以轻松地训练和比较基线算法或自定义模型，以进行节点分类，链接预测，图分类，社区发现等任务。它提供了多种图学习算法实现，包括图神经网络、图表示学习、图预训练模型、图鲁棒性方法等，并基于高效稀疏算子将模型训练速度提高多个量级。自 2020 年 4 月正式启动以来，CogDL 已经发布了若干个版本，持续不断地进行改进和优化，助力于图深度学习的研究和应用。



CogDL 核心模块

CogDL图深度学习框架



CogDL 主页

CogDL 源码

GRL: NE&GNN

Network Embedding ⇔ Matrix Factorization

Graph Neural Networks ⇨ GNN Pre-Training

Graph Data & Benchmarks

OPEN GRAPH BENCHMARK

https://alchemy.tencent.com/

Microsoft Academic

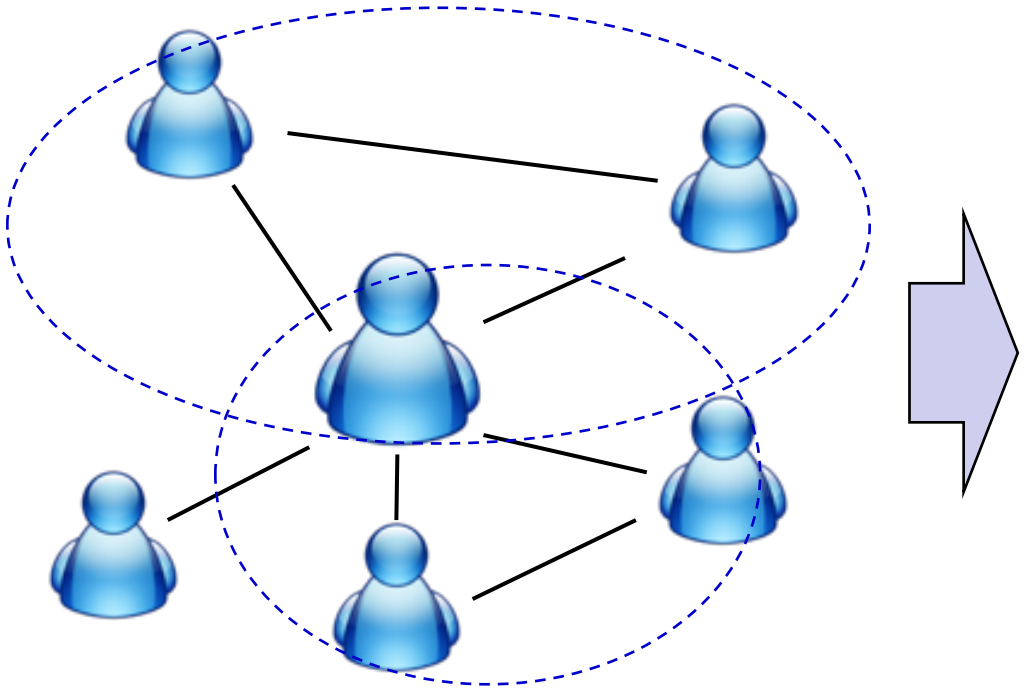
AMiner

CogDL

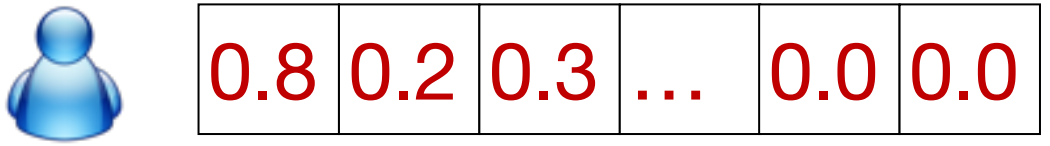
https://github.com/thudm/cogdl

Representation Learning on Networks

Representation Learning/
Graph Embedding

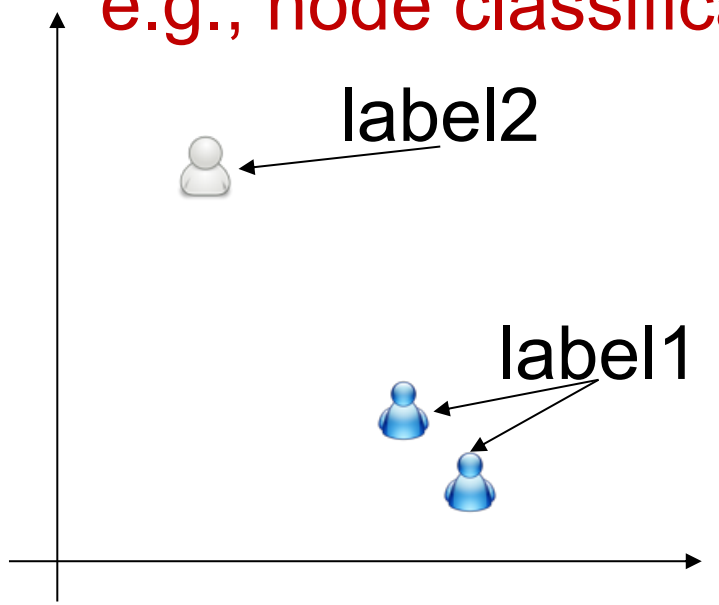


d-dimensional vector, $d << |V|$



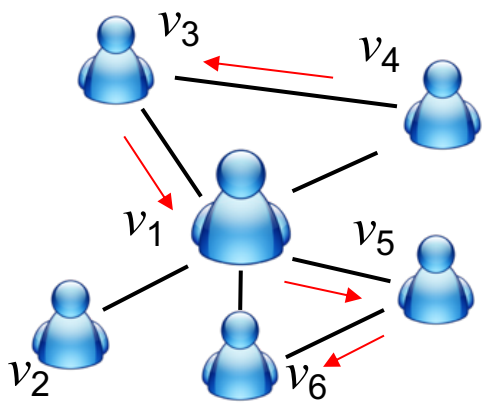
Users with the same label are located in closer

e.g., node classification



DeepWalk: Random Walk + Word2Vec

Random walk



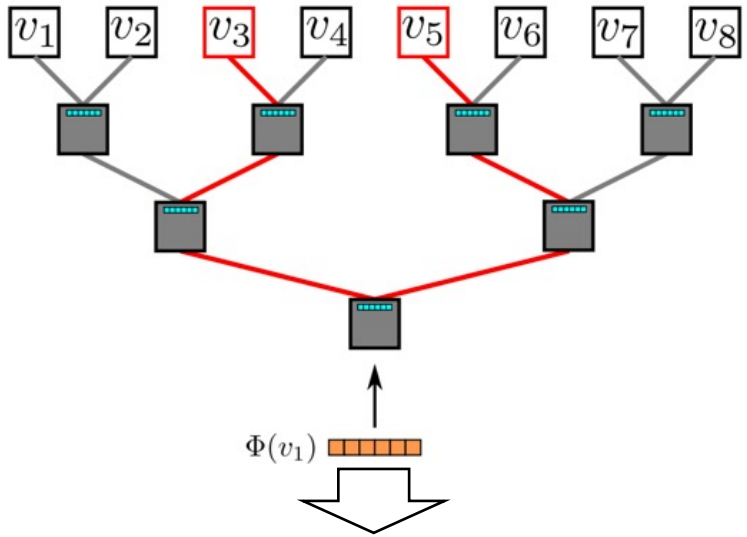
One example RW path

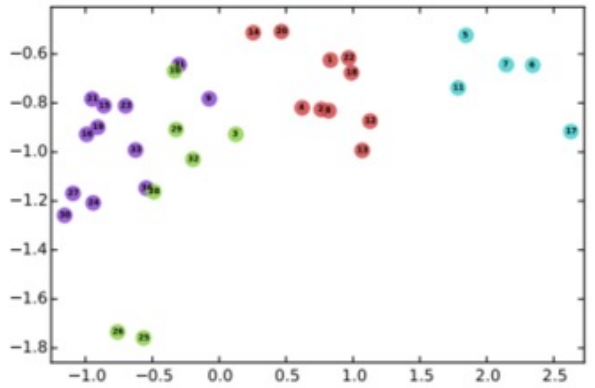
$v_4 \quad v_3 \quad v_1 \quad v_5 \quad v_6$

$\mathcal{W}_{v_4} = 4$

$u_k \begin{bmatrix} 3 \\ 1 \\ 5 \\ 1 \\ \vdots \end{bmatrix} v_j \rightarrow \Phi$ (d × j)

SkipGram with Hierarchical softmax





$$P(\{v_{i-w}, \cdots, v_{i+w}\} \backslash v_i | \Phi(v_i))) = \prod_{j=i-w, j \neq i}^{i+w} P(v_j | \Phi(v_i))$$

Hierarchical softmax

$$P(v_j | \Phi(v_i)) = \prod_{l=1}^{\log |V|} P(b_l | \Phi(v_i)) = \prod_{l=1}^{\log |V|} 1/(1 + e^{-\Phi(v_i) \cdot \Psi(b_l)})$$

1. B. Perozzi, R. Al-Rfou, and S. Skiena. 2014. Deepwalk: Online learning of social representations. *KDD*, 701–710.

Parameter Learning

- Randomly initialize the representations
- Each classifier in the hierarchy has a set of weights
- Use SGD (stochastic gradient descent) to update both classifier weights and vertex representations simultaneously

$$\mathcal{L} = \sum_{v \in V} \sum_{c \in W_v} -\log(P(c|v))$$

$$p(c|v) = \frac{\exp(\boldsymbol{z}_v^\top \boldsymbol{z}_c)}{\sum_{u \in V} \exp(\boldsymbol{z}_v^\top \boldsymbol{z}_u)}$$

Results: BlogCatalog

Name	BlogCatalog
$\|V\|$	10,312
$\|E\|$	333,983
$\|\mathcal{Y}\|$	39
Labels	Interests

	% Labeled Nodes	10%	20%	30%	40%	50%	60%	70%	80%	90%
Micro-F1(%)	DeepWalk	**36.00**	**38.20**	**39.60**	**40.30**	**41.00**	**41.30**	41.50	41.50	42.00
	SpectralClustering	31.06	34.95	37.27	38.93	39.97	40.99	**41.66**	**42.42**	**42.62**
	EdgeCluster	27.94	30.76	31.85	32.99	34.12	35.00	34.63	35.99	36.29
	Modularity	27.35	30.74	31.77	32.97	34.09	36.13	36.08	37.23	38.18
	wvRN	19.51	24.34	25.62	28.82	30.37	31.81	32.19	33.33	34.28
	Majority	16.51	16.66	16.61	16.70	16.91	16.99	16.92	16.49	17.26
Macro-F1(%)	DeepWalk	**21.30**	**23.80**	25.30	26.30	27.30	27.60	27.90	28.20	28.90
	SpectralClustering	19.14	23.57	**25.97**	**27.46**	**28.31**	**29.46**	**30.13**	**31.38**	**31.78**
	EdgeCluster	16.16	19.16	20.48	22.00	23.00	23.64	23.82	24.61	24.92
	Modularity	17.36	20.00	20.80	21.85	22.65	23.41	23.89	24.20	24.97
	wvRN	6.25	10.13	11.64	14.24	15.86	17.18	17.98	18.86	19.57
	Majority	2.52	2.55	2.52	2.58	2.58	2.63	2.61	2.48	2.62

- Feed the learned representation for node classification
- DeepWalk (node representation learning) performs well, especially when labels are sparse

Results: YouTube

Name	YouTube
$\|V\|$	1,138,499
$\|E\|$	2,990,443
$\|\mathcal{Y}\|$	47
Labels	Groups

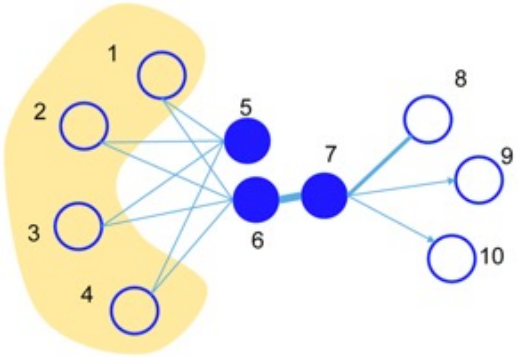
	% Labeled Nodes	1%	2%	3%	4%	5%	6%	7%	8%	9%	10%
Micro-F1(%)	DeepWalk	**37.95**	**39.28**	**40.08**	**40.78**	**41.32**	**41.72**	**42.12**	**42.48**	**42.78**	**43.05**
	SpectralClustering	—	—	—	—	—	—	—	—	—	—
	EdgeCluster	23.90	31.68	35.53	36.76	37.81	38.63	38.94	39.46	39.92	40.07
	Modularity	—	—	—	—	—	—	—	—	—	—
	wvRN	26.79	29.18	33.1	32.88	35.76	37.38	38.21	37.75	38.68	39.42
	Majority	24.90	24.84	25.25	25.23	25.22	25.33	25.31	25.34	25.38	25.38
Macro-F1(%)	DeepWalk	**29.22**	**31.83**	**33.06**	**33.90**	**34.35**	**34.66**	**34.96**	**35.22**	**35.42**	**35.67**
	SpectralClustering	—	—	—	—	—	—	—	—	—	—
	EdgeCluster	19.48	25.01	28.15	29.17	29.82	30.65	30.75	31.23	31.45	31.54
	Modularity	—	—	—	—	—	—	—	—	—	—
	wvRN	13.15	15.78	19.66	20.9	23.31	25.43	27.08	26.48	28.33	28.89
	Majority	6.12	5.86	6.21	6.1	6.07	6.19	6.17	6.16	6.18	6.19

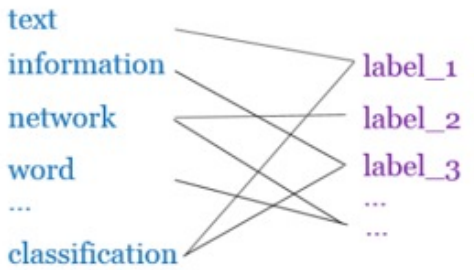
- Similar results on YouTube
- Spectral Clustering does not scale to large graphs

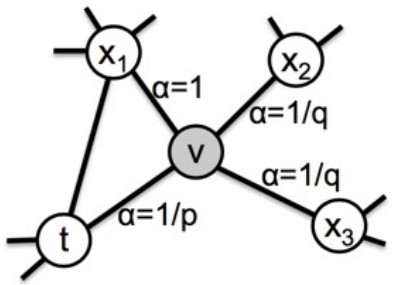
- DeepWalk utilizes fixed-length, unbiased random walks to generate context for each node, can we do better?

Later…

- LINE[1]: explicitly preserves both *first-order* and *second-order* proximities.
- PTE[2]: learn heterogeneous text network embedding via a semi-supervised manner.
- Node2vec[3]: use a biased random walk to better explore node's neighborhood.
- Metapath2vec[4]: meta-path-based random walks for heterogeneous networks.
- GATNE[5]: inductive learning for hetero-geneous networks.







1. J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei. 2015. Line: Large-scale information network embedding. *WWW'15*, 1067–1077.
2. J. Tang, M. Qu, and Q. Mei. 2015. Pte: Predictive text embedding through large-scale heterogeneous text networks. *KDD'15*, 1165–1174.
3. A. Grover and J. Leskovec. 2016. node2vec: Scalable feature learning for networks. *KDD'16*, 855–864.
4. Y. Dong, C. V. Nitesh and S. Ananthram. 2017. metapath2vec: Scalable Representation Learning for Heterogeneous Networks. KDD'14.
5. Y. Cen, X. Zou, J. Zhang, H. Yang, J. Zhou, and J. Tang. Representation Learning for Attributed Multiplex Heterogeneous Network. KDD'19.

LINE: First-order proximity

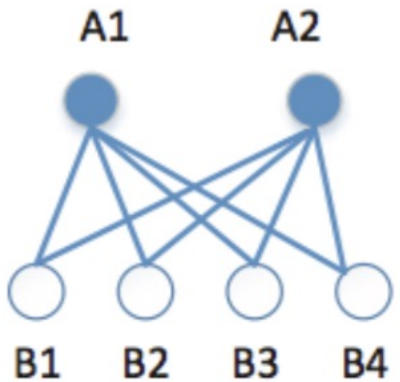
Definition

- The **first-order** proximity in a network is the **local** pairwise proximity between two vertices.
- For each pair of vertices linked by an edge (v_i, v_j), w_{ij} is the weight of the edge: indicates the first-order proximity between v_i and v_j.
- If no edge is observed between v_i and v_j, their **first-order** proximity is 0.

LINE: Second-order proximity

Definition

- The **second-order** proximity between a pair of vertices (v_i, v_j) in a network is the similarity between their neighborhood network structures.
- If no vertex is linked from/to both v_i and v_j, the **second-order** proximity between v_i and v_j is 0.



LINE: Information Network Embedding

- Given a large network $G = (V, E)$, LINE aims to represent each vertex $v \in V$ into a low-dimensional space R^d
- Goal: learning a function $f_G : V \rightarrow R^d$, $d \ll |V|$.
  In R^d, both the first-order proximity and the second-order proximity between the vertices are preserved
- For each node vi ∈ V, we use $\vec{u}$i ∈ Rd to represent the corresponding low-dimensional vector representation.

LINE with First-order Proximity

- For each undirected e_{ij}, we define the joint probability between vertex v_i and v_j as follows:

$$p_1(v_i, v_j) = \frac{1}{1 + \exp(-\vec{u}_i^T \cdot \vec{u}_j)}$$

- It defines distribution $p(\cdot, \cdot)$ over the space $V \times V$, and its empirical probability can be defined as $\hat{p}_1(i, j) = \frac{w_{i,j}}{W}$ where $W = \sum_{i,j \in E} w_{i,j}$
- Objective Function: Minimize the following **objective function**, where $d(\cdot, \cdot)$ is the distance between two distributions.

$$O_1 = d(\hat{p}_1(\cdot, \cdot), p_1(\cdot, \cdot))$$

- Instantiate $d(\cdot, \cdot)$ as KL-divergence:

$$O_1 = -\sum_{(i,j) \in E} w_{ij} \log p_1(v_i, v_j)$$

LINE with Second-order Proximity

- We first define the probability of "context" v_j generated by vertex v_i as:

$$p_2(v_j \mid v_i) = \frac{\exp(\vec{u'}_j^T \cdot \vec{u}_i)}{\sum_{k=1}^{|V|} \exp(\vec{u'}_k^T \cdot \vec{u}_i)}$$

- We minimize the following objective function:

$$O_2 = \sum_{i \in V} \lambda_i d(\hat{p}_2(\cdot \mid v_i), p_2(\cdot \mid v_i))$$

The empirical probability $\hat{p}_2(v_j|v_i)$ defined as: $\hat{p}_2(v_j|v_i) = \frac{w_{ij}}{d_i}$ where d_i is the out-degree of vertex i.

- Replacing $d(\cdot, \cdot)$ with KL-divergence, setting $\lambda_i = d_i$, we have:

$$O_2 = -\sum_{(i,j) \in E} w_{ij} \log p_2(v_j \mid v_i)$$

Model Optimization

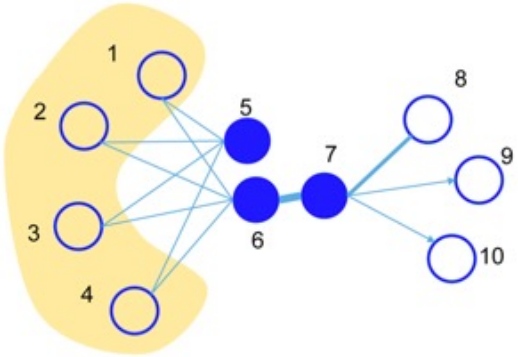
- Optimizing objective O_2 is computationally expensive.
- Adopt the approach of **negative sampling**. More specifically, it specifies the following objective function for each edge $e_{i,j}$,

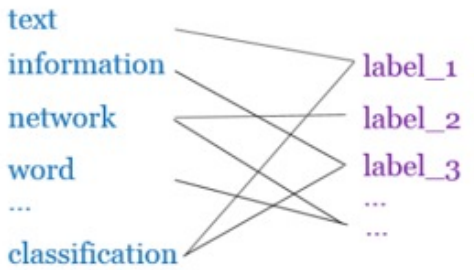
$$\underbrace{\log\sigma(\vec{u}_j'^T \cdot \vec{u}_i)}_{\text{the observed edges}} + \underbrace{\sum_{i=1}^{K} E_{v_n} \sim P_n(v)[\log\sigma(-\vec{u}_n'^T \cdot \vec{u}_i)]}_{\text{the negative edges drawn from the noise distribution}}$$

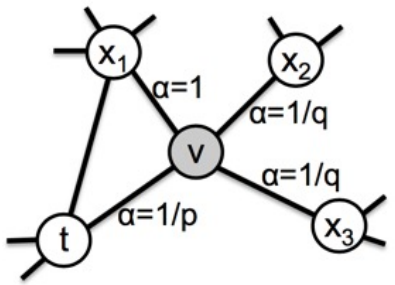
- We can use **asynchronous stochastic gradient algorithm (ASGD)** for optimizing above Eqn.

Later…

- LINE[1]: explicitly preserves both *first-order* and *second-order* proximities.
- PTE[2]: learn heterogeneous text network embedding via a semi-supervised manner.
- Node2vec[3]: use a biased random walk to better explore node's neighborhood.
- Metapath2vec[4]: meta-path-based random walks for heterogeneous networks.
- GATNE[5]: inductive learning for hetero-geneous networks.



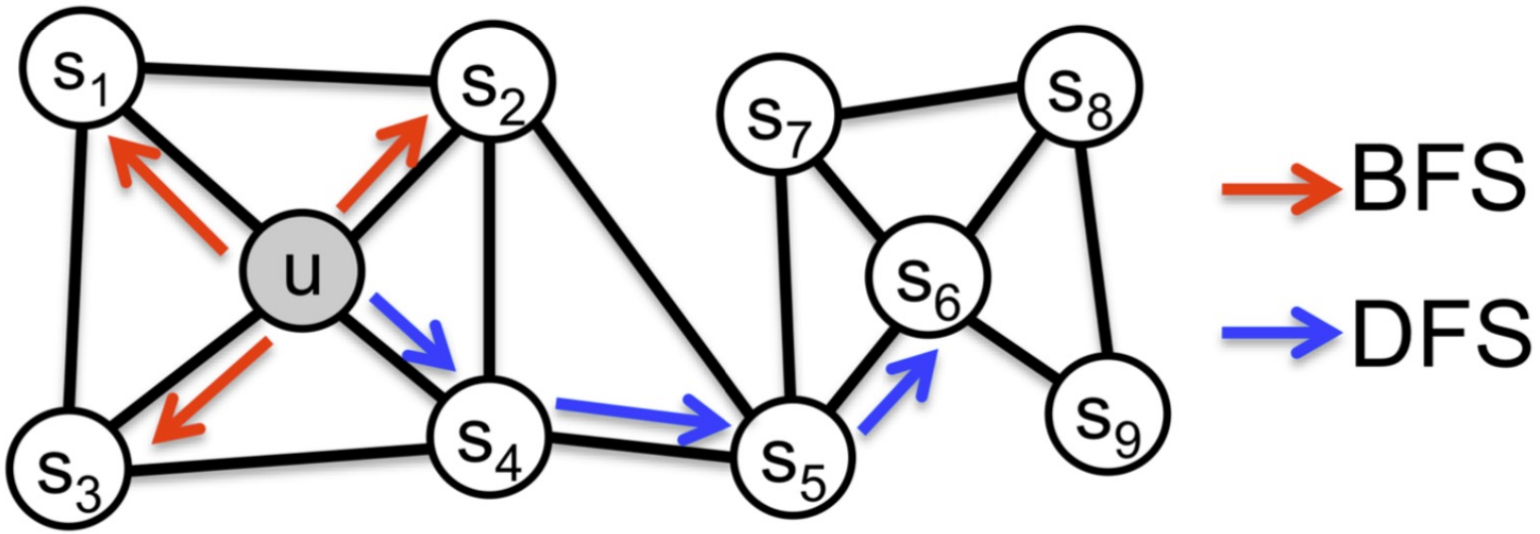




1. J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei. 2015. Line: Large-scale information network embedding. *WWW'15*, 1067–1077.
2. J. Tang, M. Qu, and Q. Mei. 2015. Pte: Predictive text embedding through large-scale heterogeneous text networks. *KDD'15*, 1165–1174.
3. A. Grover and J. Leskovec. 2016. node2vec: Scalable feature learning for networks. *KDD'16*, 855–864.
4. Y. Dong, C. V. Nitesh and S. Ananthram. 2017. metapath2vec: Scalable Representation Learning for Heterogeneous Networks. KDD'14.
5. Y. Cen, X. Zou, J. Zhang, H. Yang, J. Zhou, and J. Tang. Representation Learning for Attributed Multiplex Heterogeneous Network. KDD'19.

node2vec: Biased Walks

- Use biased random walks to trade off local and global views of the network

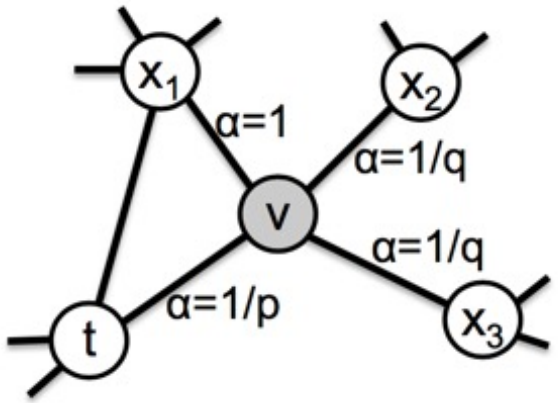


- Biased walks is a special case of random walk, thus node2vec is a special case of DeepWalk

1. A. Grover and J. Leskovec. 2016. node2vec: Scalable feature learning for networks. *KDD*, 855–864.

node2vec

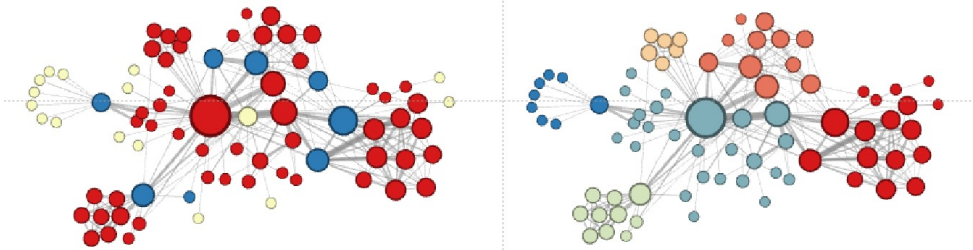
- Biased random walk R that given a node v generates random walk neighborhood $N_{rw}(v)$
- Return parameter p:
  - Return back to the previous node
- In-out parameter q:
  - Moving outwards (DFS) vs. inwards (BFS)



node2vec

- Biased random walk R that given a node v generates random walk neighborhood $N_{rw}(v)$
- Return parameter p:
  - Return back to the previous node
- In-out parameter q:
  - Moving outwards (DFS) vs. inwards (BFS)

Interactions of characters in a novel:



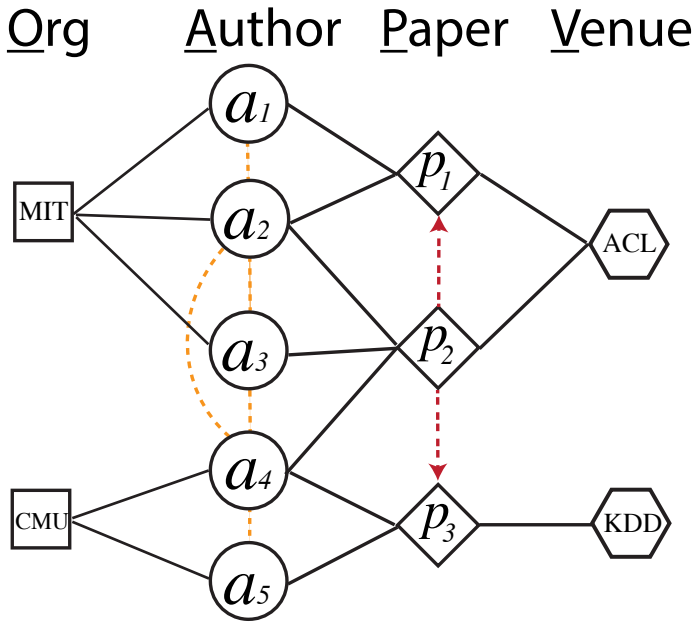
p=1, q=2
Microscopic view of the network neighbourhood

p=1, q=0.5
Macroscopic view of the network neighbourhood

Picture snipped from Leskovec

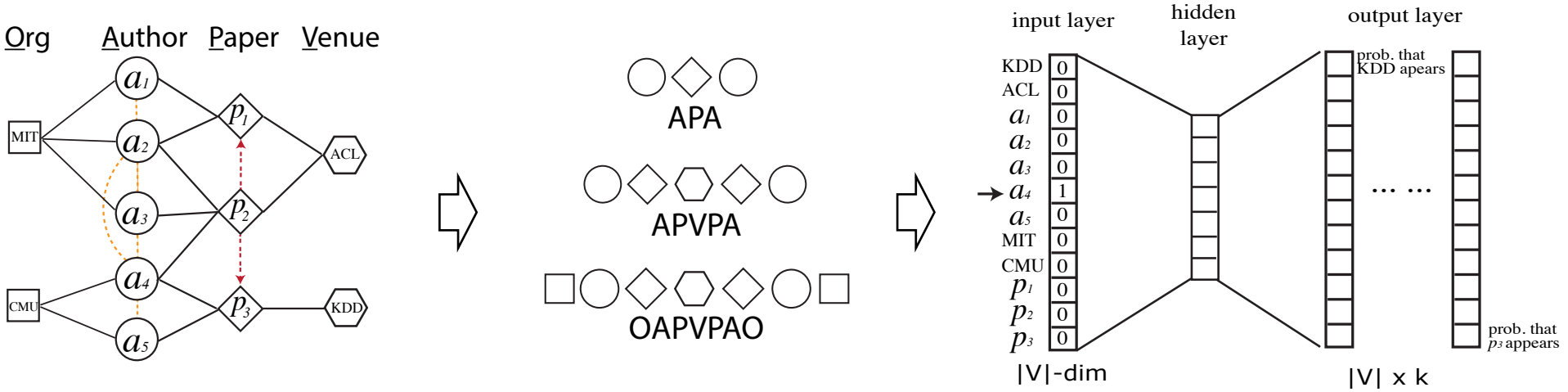
Metapath2vec: Heterogeneous Random Walk

- Input: a heterogeneous graph $G = (V, E)$
- Output: $\boldsymbol{X} \in R^{|V| \times k}, k \ll |V|$, k-dim vector $\boldsymbol{X}_v$ for each node *v*.



- How do we random walk over heterogeneous networks?
- How do we apply skip-gram over different types of nodes?

Metapath2vec: Heterogeneous Random Walk

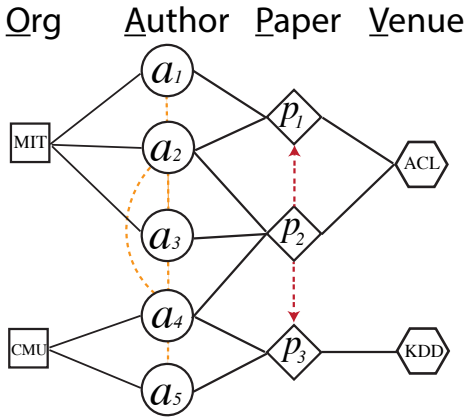


meta-path-based random walks

skip-gram

1. Sun and Han. Mining heterogeneous information networks: Principles and Methodologies. Morgan & Claypool Publishers, 2012.
2. Dong et al. metapath2vec: scalable representation learning for heterogeneous networks. In *ACM KDD* 2017. **The most cited paper in KDD'17 as of 2018.**

Metapath2vec: Heterogeneous Random Walk



- Given a meta-path scheme

$$\mathcal{P}: V_1 \xrightarrow{R_1} V_2 \xrightarrow{R_2} \cdots V_t \xrightarrow{R_t} V_{t+1} \cdots \xrightarrow{R_{l-1}} V_l$$

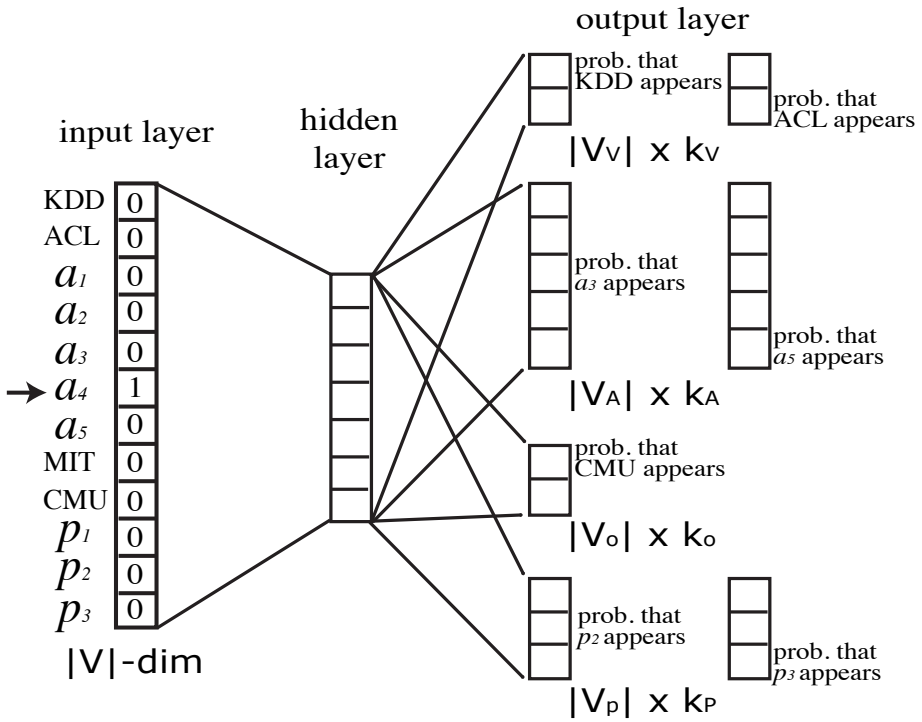
- The transition probability at step *i* is defined as

$$p(v^{i+1}|v_t^i, \mathcal{P}) = \begin{cases} \frac{1}{|N_{t+1}(v_t^i)|} & (v^{i+1}, v_t^i) \in E, \phi(v^{i+1}) = t+1 \\ 0 & (v^{i+1}, v_t^i) \in E, \phi(v^{i+1}) \neq t+1 \\ 0 & (v^{i+1}, v_t^i) \notin E \end{cases}$$

- Recursive guidance for random walkers, i.e.,

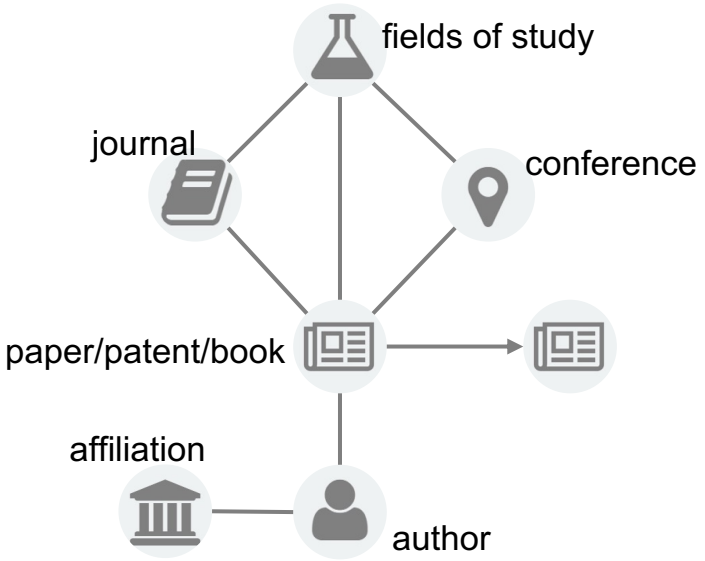
$$p(v^{i+1}|v_t^i) = p(v^{i+1}|v_1^i), \text{ if } t = l$$

Metapath2vec: Heterogeneous Random Walk



$$\mathcal{O}(\mathbf{X}) = \log \sigma(X_{c_t} \cdot X_v) + \sum_{k=1}^{K} \mathbb{E}_{u_t^k \sim P_t(u_t)} [\log \sigma(-X_{u_t^k} \cdot X_v)]$$

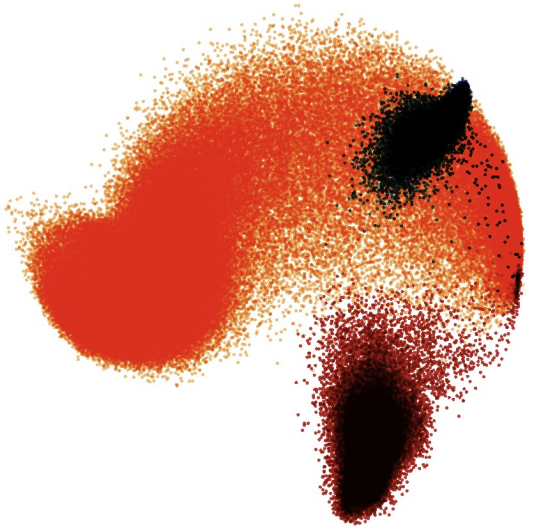
Application: Embedding Academic Graph



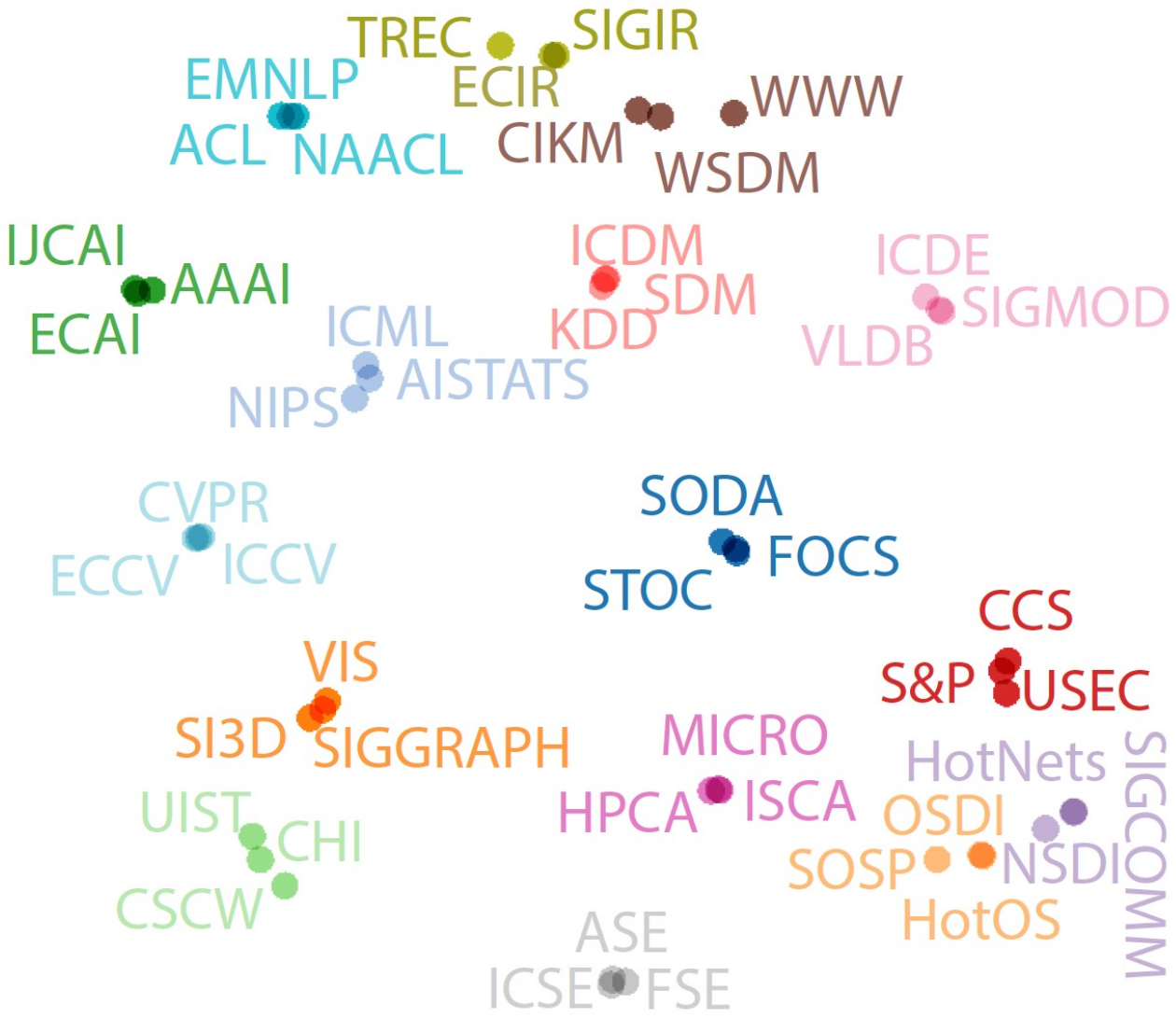
Microsoft Academic Graph
&
AMiner

metapath2vec

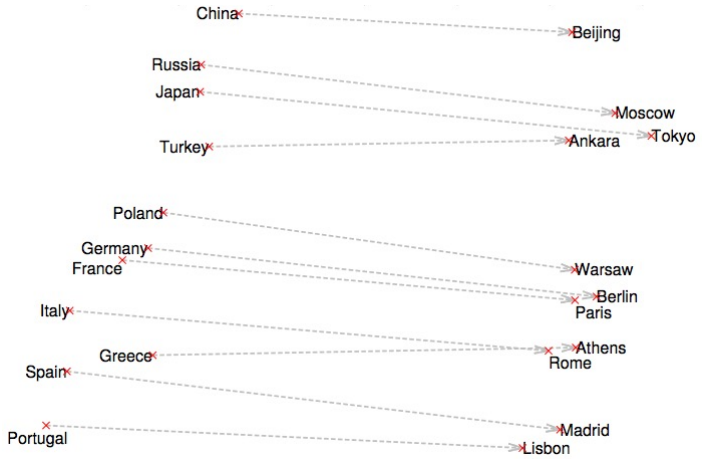
metapath2vec++



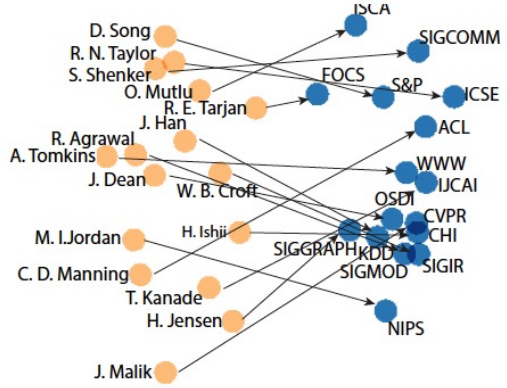
Application 2: Node Clustering



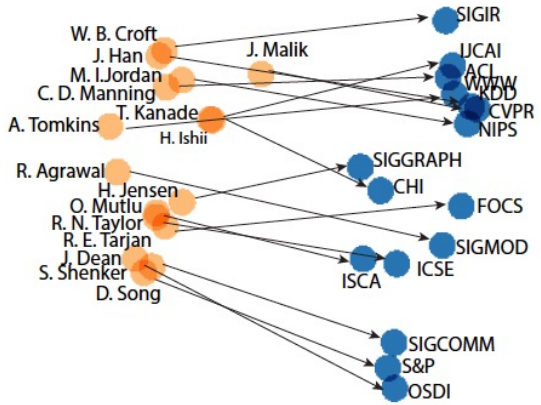
Visualization



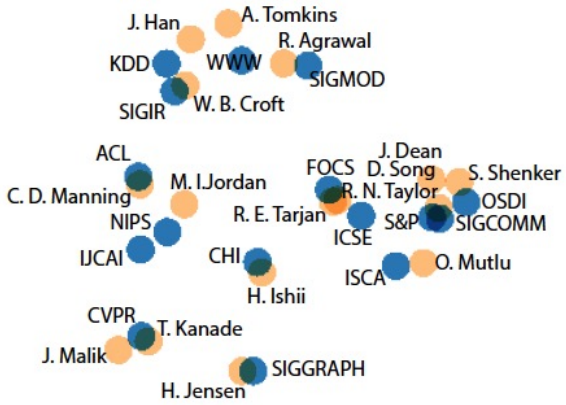
word2vec [Mikolov, 2013]



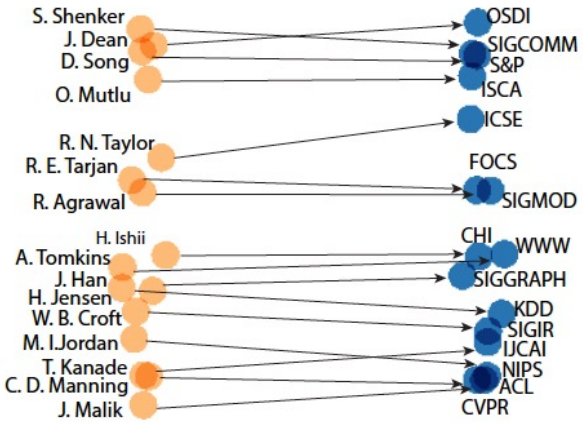
(a) DeepWalk/node2vec



(b) PTE



(c) *metapath2vec*

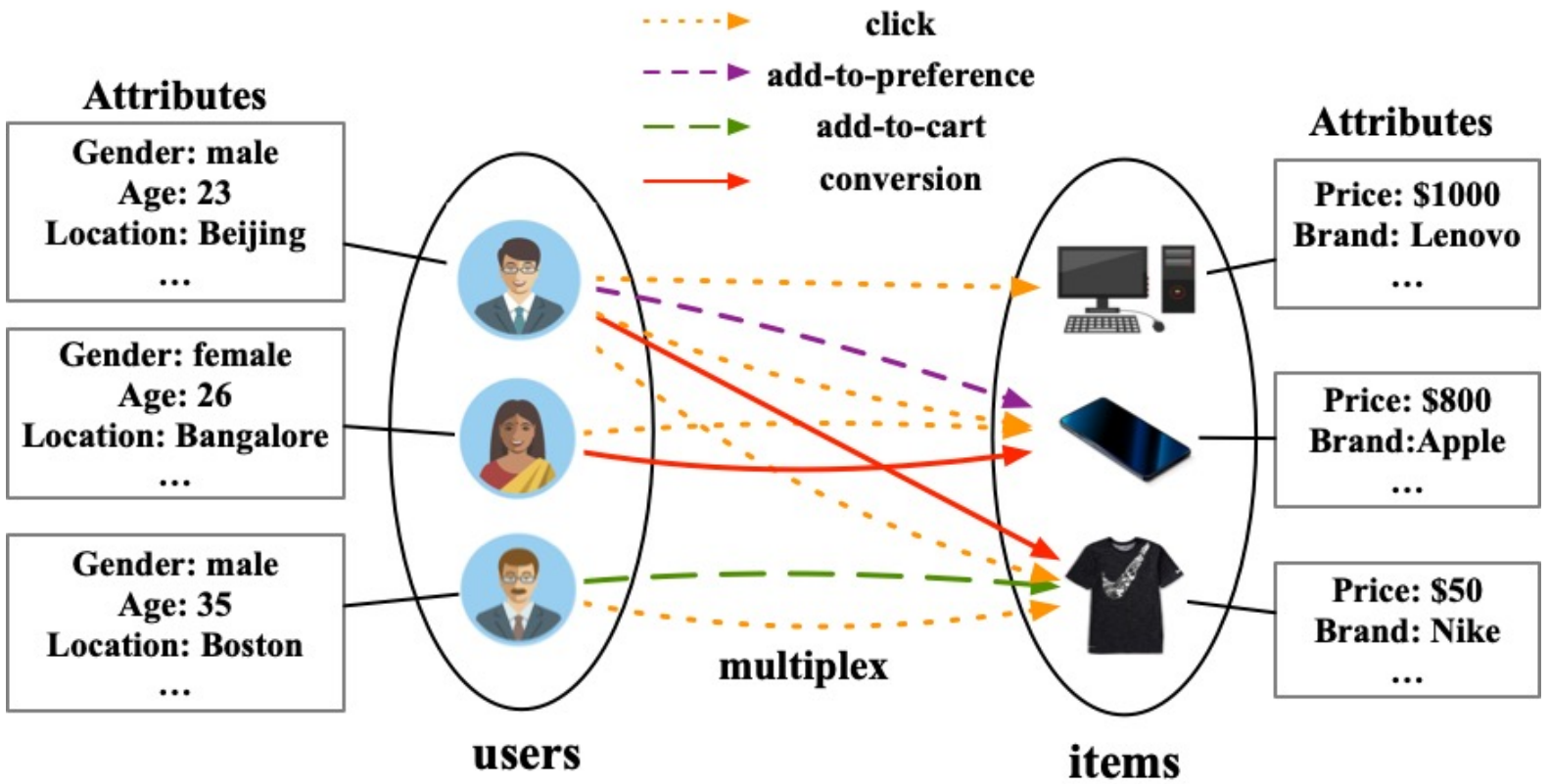


(d) *metapath2vec++*

http://projector.tensorflow.org/

GATNE: Attributed Multiplex Heterogeneous Network Embedding

- Recommend items to users by considering *Attributed Multiplex Heterogeneous Networks* (AMHEN)

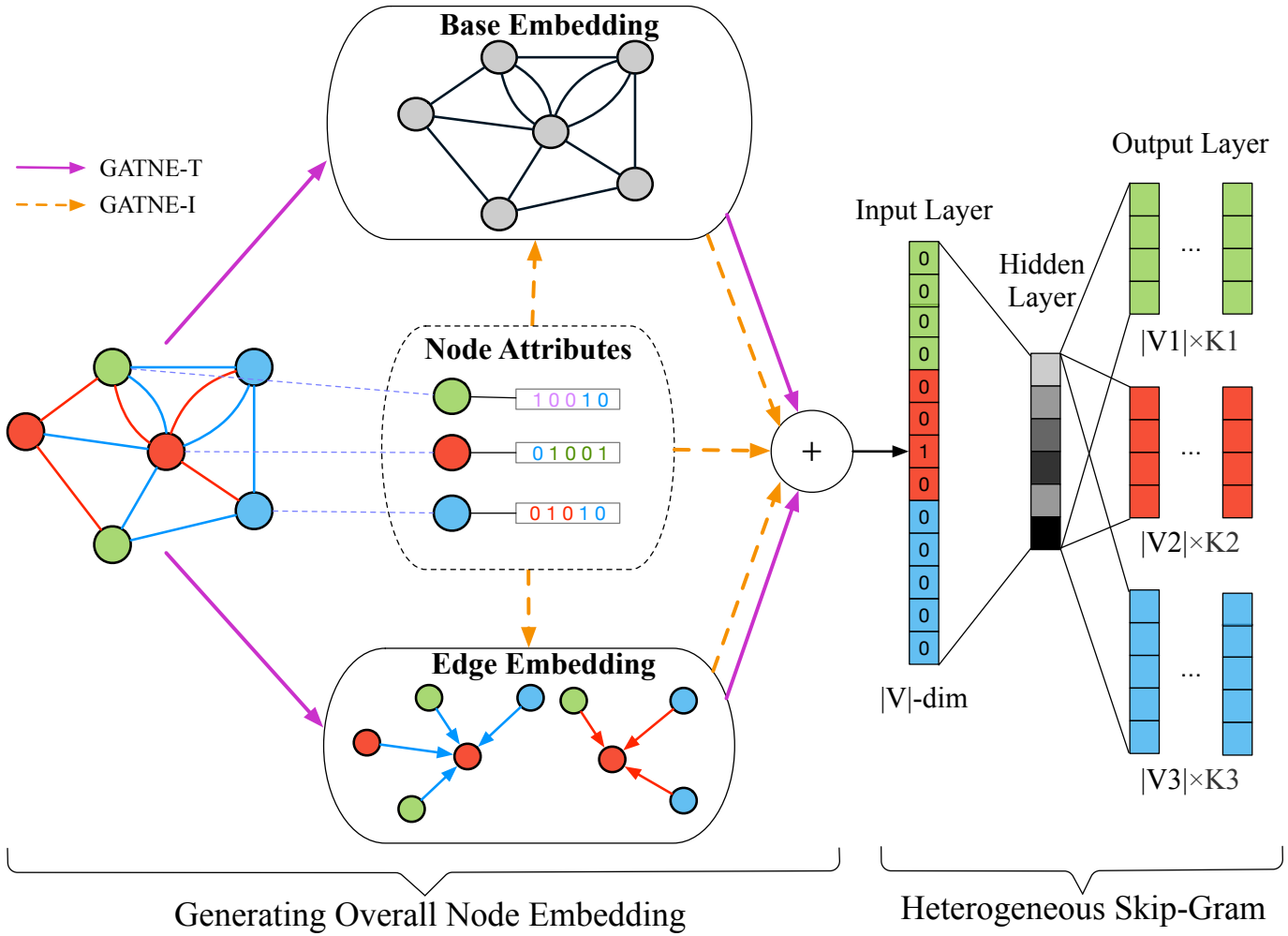


1. Y. Cen, X. Zou, J. Zhang, H. Yang, J. Zhou and J. Tang. Representation Learning for Attributed Multiplex Heterogeneous Network. KDD'19.

Different Types of Network Embedding

Network Type	Method	Heterogeneity		Attribute
		Node Type	Edge Type	
Homogeneous Network (HON)	DeepWalk [27] LINE [35] node2vec [10] NetMF [29] NetSMF [28]	Single	Single	/
Attributed Homogeneous Network (AHON)	TADW [41] LANE [16] AANE [15] SNE [20] DANE [9] ANRL [44]	Single	Single	Attributed
Heterogeneous Network (HEN)	PTE [34] metapath2vec [7] HERec [31]	Multi	Single	/
Attributed HEN (AHEN)	HNE [3]	Multi	Single	Attributed
Multiplex Heterogeneous Network (MHEN)	PMNE [22] MVE [30] MNE [43] mvn2vec [32]	Single	Multi	/
	GATNE-T	Multi	Multi	
Attributed MHEN (AMHEN)	*GATNE*-I	Multi	Multi	Attributed

Multiplex Heterogeneous Graph Embedding



1. Y. Cen, X. Zou, J. Zhang, H. Yang, J. Zhou and J. Tang. Representation Learning for Attributed Multiplex Heterogeneous Network. KDD'19.

Recommendation Results

- Data
  - Small: Amazon, YouTube, Twitter w/ 10*K* nodes
  - Large: Alibaba w/ 40M nodes and 0.5B edges

	Amazon			YouTube			Twitter			Alibaba-S		
	ROC-AUC	PR-AUC	F1	ROC-AUC	PR-AUC	F1	ROC-AUC	PR-AUC	F1	ROC-AUC	PR-AUC	F1
DeepWalk	94.20	94.03	87.38	71.11	70.04	65.52	69.42	72.58	62.68	59.39	60.62	56.10
node2vec	94.47	94.30	87.88	71.21	70.32	65.36	69.90	73.04	63.12	62.26	63.40	58.49
LINE	81.45	74.97	76.35	64.24	63.25	62.35	62.29	60.88	58.18	53.97	54.65	52.85
metapath2vec	94.15	94.01	87.48	70.98	70.02	65.34	69.35	72.61	62.70	60.94	61.40	58.25
ANRL	71.68	70.30	67.72	75.93	73.21	70.65	70.04	67.16	64.69	58.17	55.94	56.22
PMNE(n)	95.59	95.48	89.37	65.06	63.59	60.85	69.48	72.66	62.88	62.23	63.35	58.74
PMNE(r)	88.38	88.56	79.67	70.61	69.82	65.39	62.91	67.85	56.13	55.29	57.49	53.65
PMNE(c)	93.55	93.46	86.42	68.63	68.22	63.54	67.04	70.23	60.84	51.57	51.78	51.44
MVE	92.98	93.05	87.80	70.39	70.10	65.10	72.62	73.47	67.04	60.24	60.51	57.08
MNE	90.28	91.74	83.25	82.30	82.18	75.03	91.37	91.65	84.32	62.79	63.82	58.74
GATNE-T	**97.44**	**97.05**	**92.87**	**84.61**	81.93	**76.83**	**92.30**	91.77	**84.96**	66.71	67.55	62.48
GATNE-I	96.25	94.77	91.36	84.47	**82.32**	**76.83**	92.04	**91.95**	84.38	**70.87**	**71.65**	**65.54**

GATNE outperforms all sorts of baselines in the various datasets.

** Code available at https://github.com/THUDM/GATNE

Alibaba Offline A/B Tests

- GATNE-I is deployed on Alibaba's distributed cloud platform for its recommendation system. The training dataset has about 100 million users and 10 million items with 10 billion interactions between them.

- Under the framework of A/B tests, an offline test is conducted on GATNE-I, MNE and DeepWalk. The experimental goal is to maximize Hit-Rate. The results demonstrate that GATNE-I improves Hit-Rate by **3.26% and 24.26%** compared to MNE and DeepWalk respectively.

Questions

- What are the fundamentals underlying the different models?
- Can we unify the different graph embedding approaches?

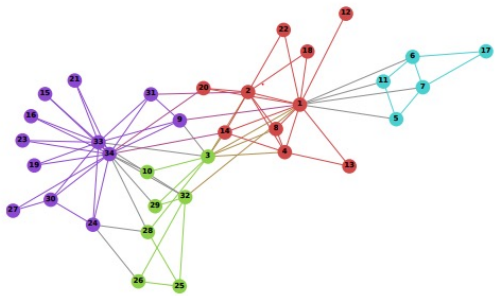
Unifying DeepWalk, LINE, PTE, and node2vec into Matrix Forms

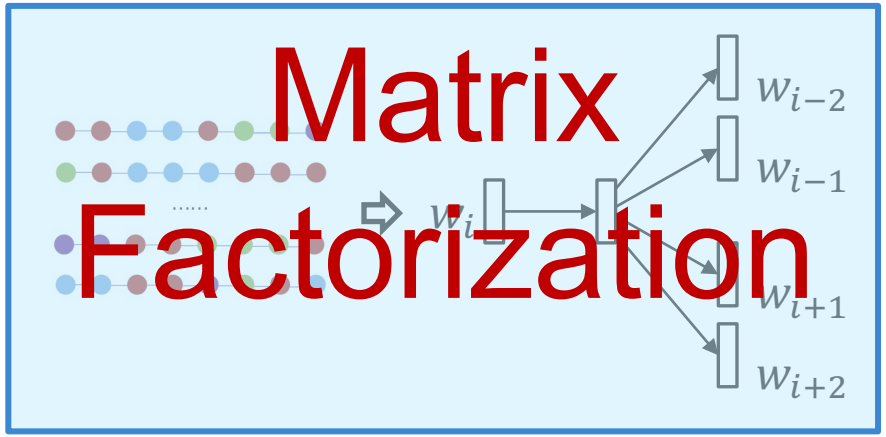
Algorithm	Closed Matrix Form
DeepWalk	$\log\left(\mathrm{vol}(G)\left(\frac{1}{T}\sum_{r=1}^{T}(\boldsymbol{D}^{-1}\boldsymbol{A})^r\right)\boldsymbol{D}^{-1}\right)-\log b$
LINE	$\log\left(\mathrm{vol}(G)\boldsymbol{D}^{-1}\boldsymbol{A}\boldsymbol{D}^{-1}\right)-\log b$
PTE	$\log\left(\begin{bmatrix}\alpha\,\mathrm{vol}(G_{\mathrm{ww}})(\boldsymbol{D}_{\mathrm{row}}^{\mathrm{ww}})^{-1}\boldsymbol{A}_{\mathrm{ww}}(\boldsymbol{D}_{\mathrm{col}}^{\mathrm{ww}})^{-1}\\ \beta\,\mathrm{vol}(G_{\mathrm{dw}})(\boldsymbol{D}_{\mathrm{row}}^{\mathrm{dw}})^{-1}\boldsymbol{A}_{\mathrm{dw}}(\boldsymbol{D}_{\mathrm{col}}^{\mathrm{dw}})^{-1}\\ \gamma\,\mathrm{vol}(G_{\mathrm{lw}})(\boldsymbol{D}_{\mathrm{row}}^{\mathrm{lw}})^{-1}\boldsymbol{A}_{\mathrm{lw}}(\boldsymbol{D}_{\mathrm{col}}^{\mathrm{lw}})^{-1}\end{bmatrix}\right)-\log b$
node2vec	$\log\left(\frac{\frac{1}{2T}\sum_{r=1}^{T}\left(\sum_u \boldsymbol{X}_{w,u}\underline{\boldsymbol{P}}^r_{c,w,u}+\sum_u \boldsymbol{X}_{c,u}\underline{\boldsymbol{P}}^r_{w,c,u}\right)}{(\sum_u \boldsymbol{X}_{w,u})(\sum_u \boldsymbol{X}_{c,u})}\right)-\log b$

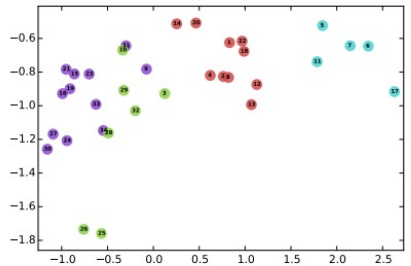
$\boldsymbol{A}$: $\boldsymbol{A}\in\mathbb{R}_{+}^{|V|\times|V|}$ is G's adjacency matrix with $\boldsymbol{A}_{i,j}$ as the edge weight between vertices i and j;
$\boldsymbol{D}_{\mathrm{col}}$: $\boldsymbol{D}_{\mathrm{col}}=\mathrm{diag}(\boldsymbol{A}^{\top}\boldsymbol{e})$ is the diagonal matrix with column sum of $\boldsymbol{A}$;
$\boldsymbol{D}_{\mathrm{row}}$: $\boldsymbol{D}_{\mathrm{row}}=\mathrm{diag}(\boldsymbol{A}\boldsymbol{e})$ is the diagonal matrix with row sum of $\boldsymbol{A}$;
$\boldsymbol{D}$: For undirected graphs ($\boldsymbol{A}^{\top}=\boldsymbol{A}$), $\boldsymbol{D}_{\mathrm{col}}=\boldsymbol{D}_{\mathrm{row}}$. For brevity, $\boldsymbol{D}$ represents both $\boldsymbol{D}_{\mathrm{col}}$ & $\boldsymbol{D}_{\mathrm{row}}$.
$\boldsymbol{D}=\mathrm{diag}(d_1,\cdots,d_{|V|})$, where d_i represents generalized degree of vertex i;
$\mathrm{vol}(G)$: $\mathrm{vol}(G)=\sum_i\sum_j \boldsymbol{A}_{i,j}=\sum_i d_i$ is the volume of an weighted graph G;
T & b: The context window size and the number of negative sampling in skip-gram, respectively.

1. Qiu et al. Network embedding as matrix factorization: unifying deepwalk, line, pte, and node2vec. *WSDM'18*. **One of the most cited papers in WSDM'18**

DeepWalk is factorizing a matrix







DeepWalk is asymptotically and implicitly factorizing

$$\log\left(\frac{\mathrm{vol}(G)}{b}\left(\frac{1}{T}\sum_{r=1}^{T}\left(\boldsymbol{D}^{-1}\boldsymbol{A}\right)^{r}\right)\boldsymbol{D}^{-1}\right)$$

$$vol(G) = \sum_i \sum_j A_{ij}$$

$\boldsymbol{A}$ Adjacency matrix

$\boldsymbol{D}$ Degree matrix

b: #negative samples

T: context window size

Skip gram with negative sampling

Skip-gram with negative sampling (SGNS)

- SGNS maintains a multiset $\mathcal{D}$ that counts the occurrence of each word-context pair (w, c)
- Objective

$$\mathcal{L} = \sum_w \sum_c (\#(w,c) \log g(x_w^T x_c) + \frac{b\#(w)\#(c)}{|\mathcal{D}|} \log g(-x_w^T x_c))$$

- For sufficiently large dimension *d*, the objective above is equivalent to factorizing the PMI matrix

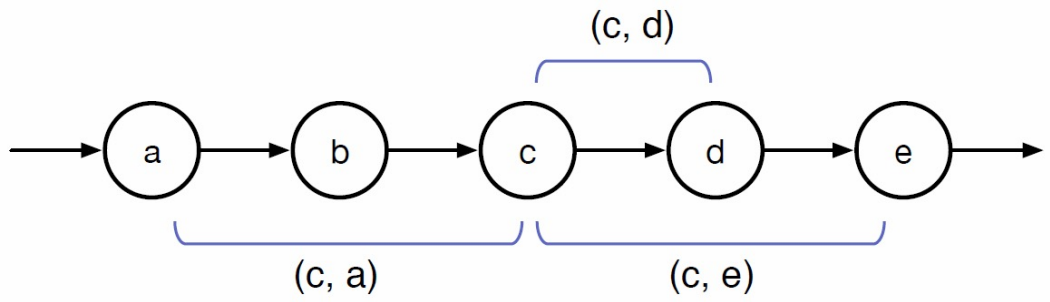
$$\log \frac{\#(w,c)|\mathcal{D}|}{b\#(w)\#(c)}$$

1. Levy and Goldberg. Neural word embeddings as implicit matrix factorization. In *NIPS 2014*

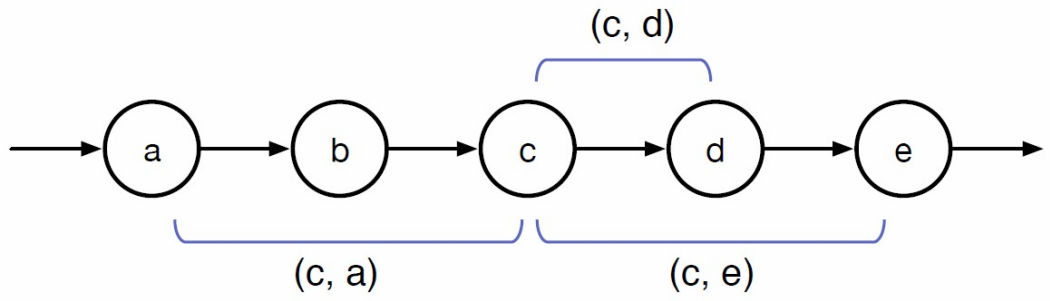
Understanding random walk + skip gram

for $n = 1, 2, \ldots, N$ **do**
    Pick w_1^n according to a probability distribution $P(w_1)$;
    Generate a vertex sequence $(w_1^n, \cdots, w_L^n)$ of length L by a random walk on network G;
    for $j = 1, 2, \ldots, L - T$ **do**
        for $r = 1, \ldots, T$ **do**
            Add vertex-context pair (w_j^n, w_{j+r}^n) to multiset $\mathcal{D}$;
            Add vertex-context pair (w_{j+r}^n, w_j^n) to multiset $\mathcal{D}$;
Run SGNS on $\mathcal{D}$ with b negative samples.

Understanding random walk + skip gram

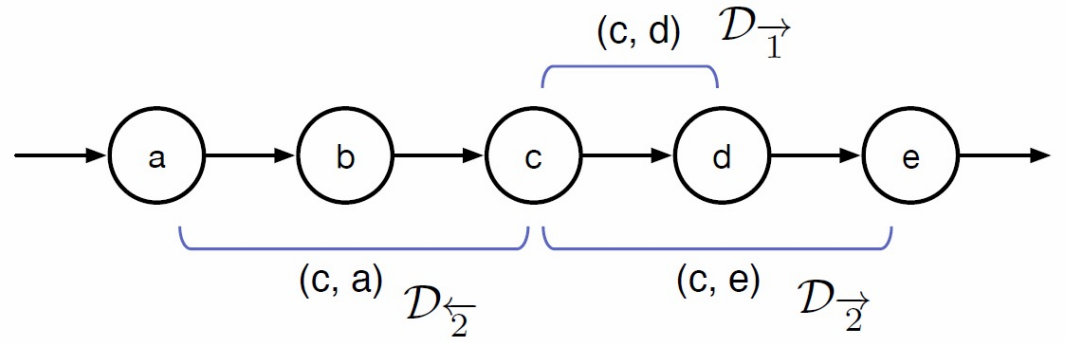


Understanding random walk + skip gram



Suppose the multiset $\mathcal{D}$ is constructed based on random walk on graphs, can we interpret $log\frac{\#(w,c)|\mathcal{D}|}{b\#(w)\#(c)}$ with graph structures?

Understanding random walk + skip gram



- Partition the multiset $\mathcal{D}$ into several sub-multisets according to the way in which each node and its context appear in a random walk node sequence.
- More formally, for $r = 1, 2, \cdots, T$, we define

$$\mathcal{D}_{\overrightarrow{r}} = \left\{(w, c) : (w, c) \in \mathcal{D}, w = w_j^n, c = w_{j+r}^n\right\}$$
$$\mathcal{D}_{\overleftarrow{r}} = \left\{(w, c) : (w, c) \in \mathcal{D}, w = w_{j+r}^n, c = w_j^n\right\}$$

Distinguish direction and distance

Understanding random walk + skip gram

$$\log\left(\frac{\#(w,c)\,|\mathcal{D}|}{b\#(w)\cdot\#(c)}\right)=\log\left(\frac{\frac{\#(w,c)}{|\mathcal{D}|}}{b\frac{\#(w)}{|\mathcal{D}|}\frac{\#(c)}{|\mathcal{D}|}}\right)$$

the length of random walk $L\to\infty$

$$\frac{\#(w,c)}{|\mathcal{D}|}\xrightarrow{p}\frac{1}{2T}\sum_{r=1}^{T}\left(\frac{d_w}{\mathrm{vol}(G)}(\boldsymbol{P}^r)_{w,c}+\frac{d_c}{\mathrm{vol}(G)}(\boldsymbol{P}^r)_{c,w}\right)$$

$$\boldsymbol{P}=\boldsymbol{D}^{-1}\boldsymbol{A}$$

$$\frac{\#(w)}{|\mathcal{D}|}\xrightarrow{p}\frac{d_w}{\mathrm{vol}(G)}\qquad\frac{\#(c)}{|\mathcal{D}|}\xrightarrow{p}\frac{d_c}{\mathrm{vol}(G)}$$

$$\frac{\#(w,c)\,|\mathcal{D}|}{\#(w)\cdot\#(c)}=\frac{\frac{\#(w,c)}{|\mathcal{D}|}}{\frac{\#(w)}{|\mathcal{D}|}\cdot\frac{\#(c)}{|\mathcal{D}|}}\xrightarrow{p}\frac{\frac{1}{2T}\sum_{r=1}^{T}\left(\frac{d_w}{\mathrm{vol}(G)}(\boldsymbol{P}^r)_{w,c}+\frac{d_c}{\mathrm{vol}(G)}(\boldsymbol{P}^r)_{c,w}\right)}{\frac{d_w}{\mathrm{vol}(G)}\cdot\frac{d_c}{\mathrm{vol}(G)}}$$

$$=\frac{\mathrm{vol}(G)}{2T}\left(\frac{1}{d_c}\sum_{r=1}^{T}(\boldsymbol{P}^r)_{w,c}+\frac{1}{d_w}\sum_{r=1}^{T}(\boldsymbol{P}^r)_{c,w}\right)$$

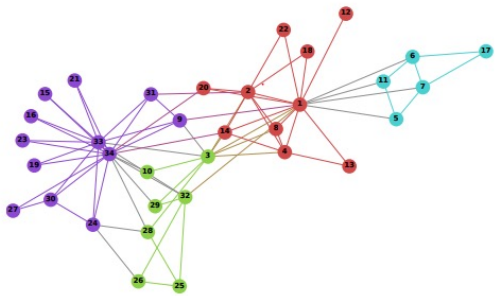
$$\frac{\#(w,c)\,|\mathcal{D}|}{\#(w)\cdot\#(c)} \xrightarrow{p} \frac{\mathrm{vol}(G)}{2T}\left(\frac{1}{d_c}\sum_{r=1}^{T}\left(\boldsymbol{P}^r\right)_{w,c}+\frac{1}{d_w}\sum_{r=1}^{T}\left(\boldsymbol{P}^r\right)_{c,w}\right)$$

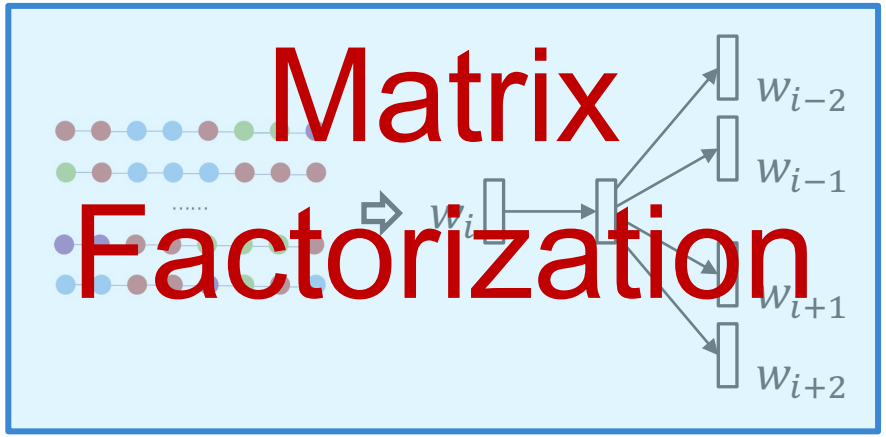
$$\frac{\mathrm{vol}(G)}{2T}\left(\sum_{r=1}^{T}\boldsymbol{P}^r\boldsymbol{D}^{-1}+\sum_{r=1}^{T}\boldsymbol{D}^{-1}\left(\boldsymbol{P}^r\right)^{\top}\right)$$

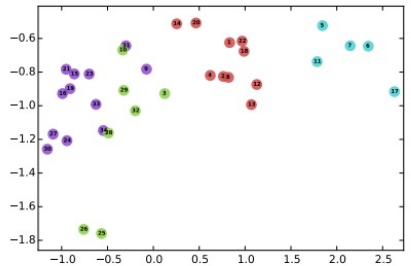
$$=\frac{\mathrm{vol}(G)}{2T}\left(\sum_{r=1}^{T}\underbrace{\boldsymbol{D}^{-1}\boldsymbol{A}\times\cdots\times\boldsymbol{D}^{-1}\boldsymbol{A}}_{r\text{ terms}}\boldsymbol{D}^{-1}+\sum_{r=1}^{T}\boldsymbol{D}^{-1}\underbrace{\boldsymbol{A}\boldsymbol{D}^{-1}\times\cdots\times\boldsymbol{A}\boldsymbol{D}^{-1}}_{r\text{ terms}}\right)$$

$$=\frac{\mathrm{vol}(G)}{T}\sum_{r=1}^{T}\underbrace{\boldsymbol{D}^{-1}\boldsymbol{A}\times\cdots\times\boldsymbol{D}^{-1}\boldsymbol{A}}_{r\text{ terms}}\boldsymbol{D}^{-1}=\mathrm{vol}(G)\left(\frac{1}{T}\sum_{r=1}^{T}\boldsymbol{P}^r\right)\boldsymbol{D}^{-1}.$$

DeepWalk is factorizing a matrix







DeepWalk is asymptotically and implicitly factorizing

$$\log\left(\frac{\mathrm{vol}(G)}{b}\left(\frac{1}{T}\sum_{r=1}^{T}\left(\boldsymbol{D}^{-1}\boldsymbol{A}\right)^{r}\right)\boldsymbol{D}^{-1}\right)$$

$$vol(G) = \sum_i \sum_j A_{ij}$$

$\boldsymbol{A}$ Adjacency matrix

$\boldsymbol{D}$ Degree matrix

b: #negative samples

T: context window size

LINE

- Objective of LINE:

$$\mathcal{L} = \sum_{i=1}^{|V|} \sum_{j=1}^{|V|} \left(\boldsymbol{A}_{i,j} \log g\left(\boldsymbol{x}_i^\top \boldsymbol{y}_j\right) + \frac{b d_i d_j}{\mathrm{vol}(G)} \log g\left(-\boldsymbol{x}_i^\top \boldsymbol{y}_j\right) \right).$$

- Align it with the Objective of SGNS:

$$\mathcal{L} = \sum_{w} \sum_{c} \left(\#(w,c) \log g\left(\boldsymbol{x}_w^\top \boldsymbol{y}_c\right) + \frac{b \#(w) \#(c)}{|\mathcal{D}|} \log g\left(-\boldsymbol{x}_w^\top \boldsymbol{y}_c\right) \right).$$

- LINE is actually factorizing

$$\log \left(\frac{\mathrm{vol}(G)}{b} \boldsymbol{D}^{-1} \boldsymbol{A} \boldsymbol{D}^{-1} \right)$$

- Recall DeepWalk's matrix form:

$$\log \left(\frac{\mathrm{vol}(G)}{b} \left(\frac{1}{T} \sum_{r=1}^{T} \left(\boldsymbol{D}^{-1} \boldsymbol{A} \right)^r \right) \boldsymbol{D}^{-1} \right).$$

Observation LINE is a special case of DeepWalk ($T = 1$).

PTE

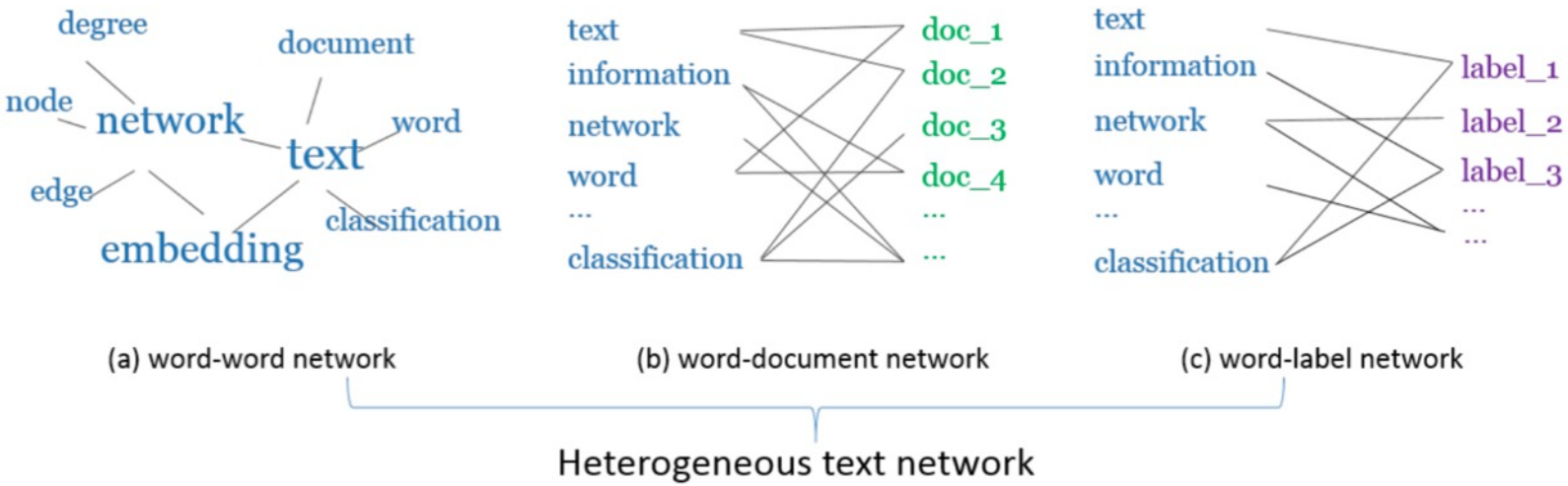


Figure 2: Heterogeneous Text Network.

$$\log\left(\begin{bmatrix}\alpha\,\mathrm{vol}(G_{\mathsf{ww}})(\boldsymbol{D}_{\mathsf{row}}^{\mathsf{ww}})^{-1}\boldsymbol{A}_{\mathsf{ww}}(\boldsymbol{D}_{\mathsf{col}}^{\mathsf{ww}})^{-1}\\ \beta\,\mathrm{vol}(G_{\mathsf{dw}})(\boldsymbol{D}_{\mathsf{row}}^{\mathsf{dw}})^{-1}\boldsymbol{A}_{\mathsf{dw}}(\boldsymbol{D}_{\mathsf{col}}^{\mathsf{dw}})^{-1}\\ \gamma\,\mathrm{vol}(G_{\mathsf{lw}})(\boldsymbol{D}_{\mathsf{row}}^{\mathsf{lw}})^{-1}\boldsymbol{A}_{\mathsf{lw}}(\boldsymbol{D}_{\mathsf{col}}^{\mathsf{lw}})^{-1}\end{bmatrix}\right)-\log b,$$

Understanding node2vec

$$\underline{\boldsymbol{T}}_{u,v,w} = \begin{cases} \frac{1}{p} & (u,v) \in E, (v,w) \in E, u = w; \\ 1 & (u,v) \in E, (v,w) \in E, u \neq w, (w,u) \in E; \\ \frac{1}{q} & (u,v) \in E, (v,w) \in E, u \neq w, (w,u) \notin E; \\ 0 & \text{otherwise.} \end{cases}$$

$$\underline{\boldsymbol{P}}_{u,v,w} = \text{Prob}\left(w_{j+1} = u | w_j = v, w_{j-1} = w\right) = \frac{\underline{\boldsymbol{T}}_{u,v,w}}{\sum_u \underline{\boldsymbol{T}}_{u,v,w}}.$$

Stationary Distribution

$$\sum_w \underline{\boldsymbol{P}}_{u,v,w} \boldsymbol{X}_{v,w} = \boldsymbol{X}_{u,v}$$

Existence guaranteed by Perron-Frobenius theorem, but may not be unique.

A. Grover and J. Leskovec. 2016. node2vec: Scalable feature learning for networks. *KDD*, 855–864.

Understanding node2vec

Theorem

node2vec is asymptotically and implicitly factorizing a matrix whose entry at w-th row, c-th column is

$$\log\left(\frac{\frac{1}{2T}\sum_{r=1}^{T}\left(\sum_u \boldsymbol{X}_{w,u}\underline{\boldsymbol{P}}^r_{c,w,u}+\sum_u \boldsymbol{X}_{c,u}\underline{\boldsymbol{P}}^r_{w,c,u}\right)}{b\left(\sum_u \boldsymbol{X}_{w,u}\right)\left(\sum_u \boldsymbol{X}_{c,u}\right)}\right)$$

1. A. Grover and J. Leskovec. 2016. node2vec: Scalable feature learning for networks. *KDD*, 855–864.

Can we directly factorize the derived matrices for learning embeddings?

NetMF

- DeepWalk is implicitly factorizing

$$\boldsymbol{M} = \log\left(\frac{\mathrm{vol}(G)}{b}\left(\frac{1}{T}\sum_{r=1}^{T}\left(\boldsymbol{D}^{-1}\boldsymbol{A}\right)^{r}\right)\boldsymbol{D}^{-1}\right)$$

- NetMF is explicitly factorizing

$$\boldsymbol{M} = \log\left(\frac{\mathrm{vol}(G)}{b}\left(\frac{1}{T}\sum_{r=1}^{T}\left(\boldsymbol{D}^{-1}\boldsymbol{A}\right)^{r}\right)\boldsymbol{D}^{-1}\right)$$

NetMF

- DeepWalk is implicitly factorizing

$$\boldsymbol{M} = \log\left(\frac{\mathrm{vol}(G)}{b}\left(\frac{1}{T}\sum_{r=1}^{T}\left(\boldsymbol{D}^{-1}\boldsymbol{A}\right)^{r}\right)\boldsymbol{D}^{-1}\right)$$

Recall that in random walk + skip-gram based embedding models: $\boldsymbol{z}_v^{\top}\boldsymbol{z}_c$ → the probability that node v and context c appear on a random walk path

- NetMF is explicitly factorizing

$$\boldsymbol{M} = \log\left(\frac{\mathrm{vol}(G)}{b}\left(\frac{1}{T}\sum_{r=1}^{T}\left(\boldsymbol{D}^{-1}\boldsymbol{A}\right)^{r}\right)\boldsymbol{D}^{-1}\right)$$

$\boldsymbol{z}_v^{\top}\boldsymbol{z}_c$ → the similarity score $\boldsymbol{M}_{vc}$ between node v and context c defined by this matrix

NetMF

1 Eigen-decomposition $\boldsymbol{D}^{-1/2}\boldsymbol{A}\boldsymbol{D}^{-1/2} \approx \boldsymbol{U}_h\boldsymbol{\Lambda}_h\boldsymbol{U}_h^\top$;

2 Approximate $\boldsymbol{M}$ with

$$\hat{\boldsymbol{M}} = \frac{\mathrm{vol}(G)}{b}\boldsymbol{D}^{-1/2}\boldsymbol{U}_h\left(\frac{1}{T}\sum_{r=1}^{T}\boldsymbol{\Lambda}_h^r\right)\boldsymbol{U}_h^\top\boldsymbol{D}^{-1/2};$$

3 Compute $\hat{\boldsymbol{M}}' = \max(\hat{\boldsymbol{M}}, 1)$;

4 Rank-d approximation by SVD: $\log\hat{\boldsymbol{M}}' = \boldsymbol{U}_d\boldsymbol{\Sigma}_d\boldsymbol{V}_d^\top$;

5 **return** $\boldsymbol{U}_d\sqrt{\boldsymbol{\Sigma}_d}$ as network embedding.

Approximate $\boldsymbol{D}^{-1/2}\boldsymbol{A}\boldsymbol{D}^{-1/2}$ with its top-h eigenpairs $\boldsymbol{U}_h\boldsymbol{\Lambda}_h\boldsymbol{U}_h^\top$

The Arnoldi algorithm [1] for significant time reduction

1. R. Lehoucq, D. Sorensen, and C.Yang. 1998. ARPACK users' guide: solution of large-scale eigenvalue problems with implicitly restarted Arnoldi methods. SIAM.

Error Bound for NetMF for a large window size *T*

- According to Frobenius norm's property

$$\left|\log \boldsymbol{M}'_{i,j} - \log \hat{\boldsymbol{M}}'_{i,j}\right| = \log \frac{\hat{\boldsymbol{M}}'_{i,j}}{\boldsymbol{M}'_{i,j}} = \log\left(1 + \frac{\hat{\boldsymbol{M}}'_{i,j} - \boldsymbol{M}'_{i,j}}{\boldsymbol{M}'_{i,j}}\right)$$

$$\leq \frac{\hat{\boldsymbol{M}}'_{i,j} - \boldsymbol{M}'_{i,j}}{\boldsymbol{M}'_{i,j}} \leq \hat{\boldsymbol{M}}'_{i,j} - \boldsymbol{M}'_{i,j} = \left|\hat{\boldsymbol{M}}'_{i,j} - \boldsymbol{M}'_{i,j}\right|$$

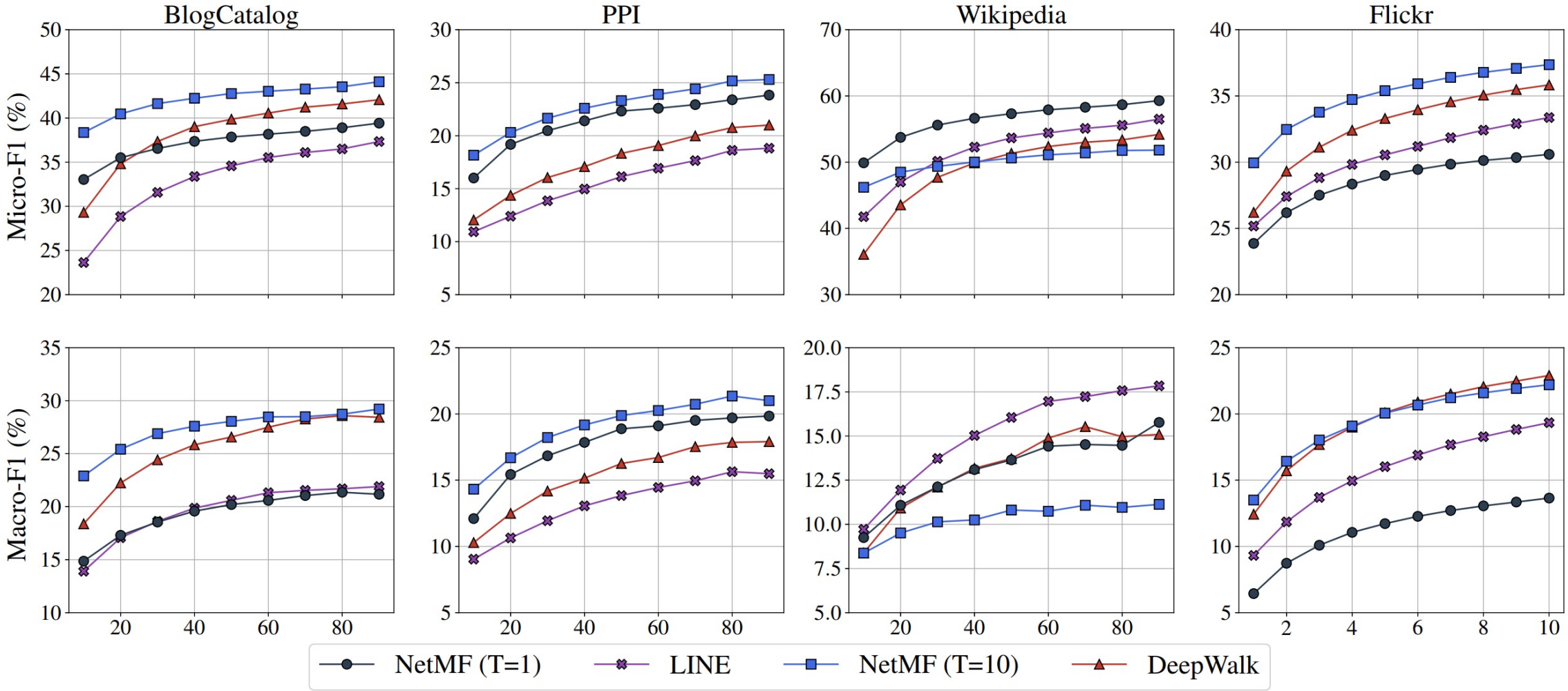
- and because $M'_{i,j}=\max(M_{i,j}, 1)>=1$, we have

$$\left|\boldsymbol{M}'_{i,j} - \hat{\boldsymbol{M}}'_{i,j}\right| = \left|\max(\boldsymbol{M}_{i,j}, 1) - \max(\hat{\boldsymbol{M}}_{i,j}, 1)\right| \leq \left|\boldsymbol{M}_{i,j} - \hat{\boldsymbol{M}}_{i,j}\right|$$

- Also because the property of NGL,

$$\sigma_s\left(\left(\frac{1}{T}\sum_{r=1}^{T}\boldsymbol{P}^r\right)\boldsymbol{D}^{-1}\right) \leq \frac{\left|\frac{1}{T}\sum_{r=1}^{T}\lambda_{p_s}^r\right|}{d_{q_1}} \Rightarrow \left\|\boldsymbol{M} - \hat{\boldsymbol{M}}\right\|_F \leq \frac{\mathrm{vol}(G)}{b d_{\min}} \sqrt{\sum_{j=k+1}^{n}\left|\frac{1}{T}\sum_{r=1}^{T}\lambda_j^r\right|^2}$$

Experimental Results



Predictive performance on varying the ratio of training data;

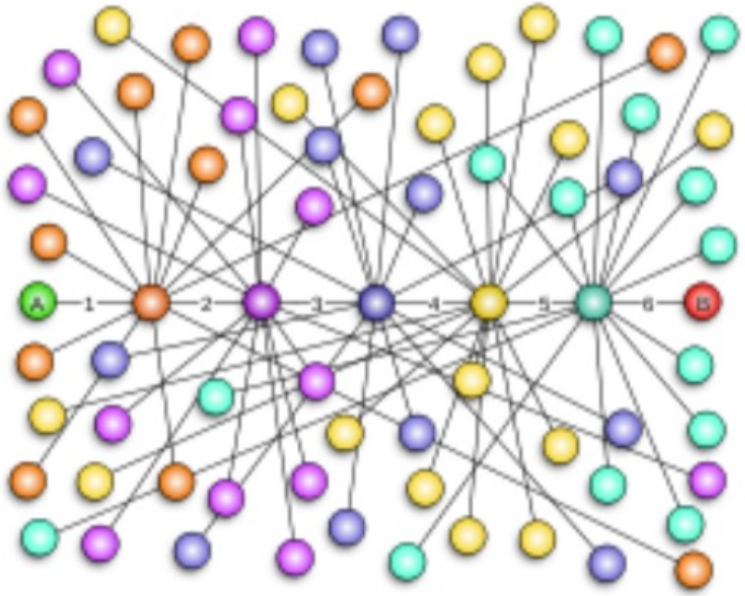
The *x*-axis represents the ratio of labeled data (%)

Network Embedding as Matrix Factorization

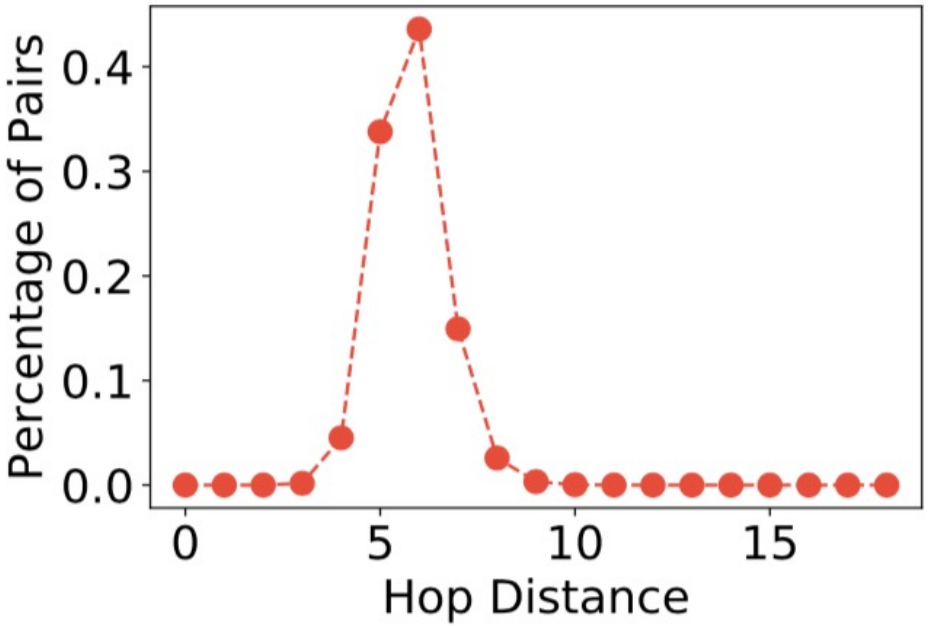
- DeepWalk $$\log\left(\frac{\mathrm{vol}(G)}{b}\left(\frac{1}{T}\sum_{r=1}^{T}\left(\boldsymbol{D}^{-1}\boldsymbol{A}\right)^{r}\right)\boldsymbol{D}^{-1}\right)$$
- LINE $$\log\left(\frac{\mathrm{vol}(G)}{b}\boldsymbol{D}^{-1}\boldsymbol{A}\boldsymbol{D}^{-1}\right)$$
- PTE $$\log\left(\begin{bmatrix}\alpha\,\mathrm{vol}(G_{\mathsf{ww}})(\boldsymbol{D}_{\mathsf{row}}^{\mathsf{ww}})^{-1}\boldsymbol{A}_{\mathsf{ww}}(\boldsymbol{D}_{\mathsf{col}}^{\mathsf{ww}})^{-1}\\ \beta\,\mathrm{vol}(G_{\mathsf{dw}})(\boldsymbol{D}_{\mathsf{row}}^{\mathsf{dw}})^{-1}\boldsymbol{A}_{\mathsf{dw}}(\boldsymbol{D}_{\mathsf{col}}^{\mathsf{dw}})^{-1}\\ \gamma\,\mathrm{vol}(G_{\mathsf{lw}})(\boldsymbol{D}_{\mathsf{row}}^{\mathsf{lw}})^{-1}\boldsymbol{A}_{\mathsf{lw}}(\boldsymbol{D}_{\mathsf{col}}^{\mathsf{lw}})^{-1}\end{bmatrix}\right)-\log b$$
- node2vec $$\log\left(\frac{\frac{1}{2T}\sum_{r=1}^{T}\left(\sum_{u}\boldsymbol{X}_{w,u}\underline{\boldsymbol{P}}_{c,w,u}^{r}+\sum_{u}\boldsymbol{X}_{c,u}\underline{\boldsymbol{P}}_{w,c,u}^{r}\right)}{b\left(\sum_{u}\boldsymbol{X}_{w,u}\right)\left(\sum_{u}\boldsymbol{X}_{c,u}\right)}\right)$$

1. Qiu et al. Network embedding as matrix factorization: unifying deepwalk, line, pte, and node2vec. *WSDM'18*. **One of the most cited papers in WSDM'18**

Challenge in NetMF



Small world



Academic graph

$$\log^{\circ}\left(\frac{\mathrm{vol}(G)}{b}\left(\frac{1}{T}\sum_{r=1}^{T}\left(\boldsymbol{D}^{-1}\boldsymbol{A}\right)^{r}\right)\boldsymbol{D}^{-1}\right)$$ is always a dense matrix.

Sparsify $\boldsymbol{S}$

For random-walk matrix polynomial $\boldsymbol{L} = \boldsymbol{D} - \sum_{r=1}^{T} \alpha_r \boldsymbol{D} \left(\boldsymbol{D}^{-1}\boldsymbol{A}\right)^r$

where $\sum_{r=1}^{T} \alpha_r = 1$ and α_r non-negative

One can construct a $(1+\epsilon)$-spectral sparsifier $\tilde{\boldsymbol{L}}$ with $O(n \log n \epsilon^{-2})$ non-zeros

in time $O(T^2 m \epsilon^{-2} \log^2 n)$

$O(T^2 m \epsilon^{-2} \log n)$ for undirected graphs

1. D. Cheng, Y. Cheng, Y. Liu, R. Peng, and S.H. Teng, Efficient Sampling for Gaussian Graphical Models via Spectral Sparsification, COLT 2015.
2. D. Cheng, Y. Cheng, Y. Liu, R. Peng, and S.H. Teng. Spectral sparsification of random-walk matrix polynomials. arXiv:1502.03496.

Sparsify $\boldsymbol{S}$

For random-walk matrix polynomial $\boldsymbol{L} = \boldsymbol{D} - \sum_{r=1}^{T} \alpha_r \boldsymbol{D} \left(\boldsymbol{D}^{-1}\boldsymbol{A}\right)^r$

where $\sum_{r=1}^{T} \alpha_r = 1$ and α_r non-negative

One can construct a $(1+\epsilon)$-spectral sparsifier $\tilde{\boldsymbol{L}}$ with $O(n \log n \epsilon^{-2})$ non-zeros

in time $O(T^2 m \epsilon^{-2} \log^2 n)$

Suppose $G = (V, E, \boldsymbol{A})$ and $\widetilde{G} = (V, \widetilde{E}, \widetilde{\boldsymbol{A}})$ are two weighted undirected networks. Let $\boldsymbol{L} = \boldsymbol{D}_G - \boldsymbol{A}$ and $\widetilde{\boldsymbol{L}} = \boldsymbol{D}_{\widetilde{G}} - \widetilde{\boldsymbol{A}}$ be their Laplacian matrices, respectively. We define G and $\widetilde{G}$ are $(1+\epsilon)$-spectrally similar if

$$\forall \boldsymbol{x} \in \mathbb{R}^n, (1-\epsilon) \cdot \boldsymbol{x}^\top \widetilde{\boldsymbol{L}} \boldsymbol{x} \leq \boldsymbol{x}^\top \boldsymbol{L} \boldsymbol{x} \leq (1+\epsilon) \cdot \boldsymbol{x}^\top \widetilde{\boldsymbol{L}} \boldsymbol{x}.$$

1. D. Cheng, Y. Cheng, Y. Liu, R. Peng, and S.H. Teng, Efficient Sampling for Gaussian Graphical Models via Spectral Sparsification, COLT 2015.
2. D. Cheng, Y. Cheng, Y. Liu, R. Peng, and S.H. Teng. Spectral sparsification of random-walk matrix polynomials. arXiv:1502.03496.

NetSMF --- Sparse

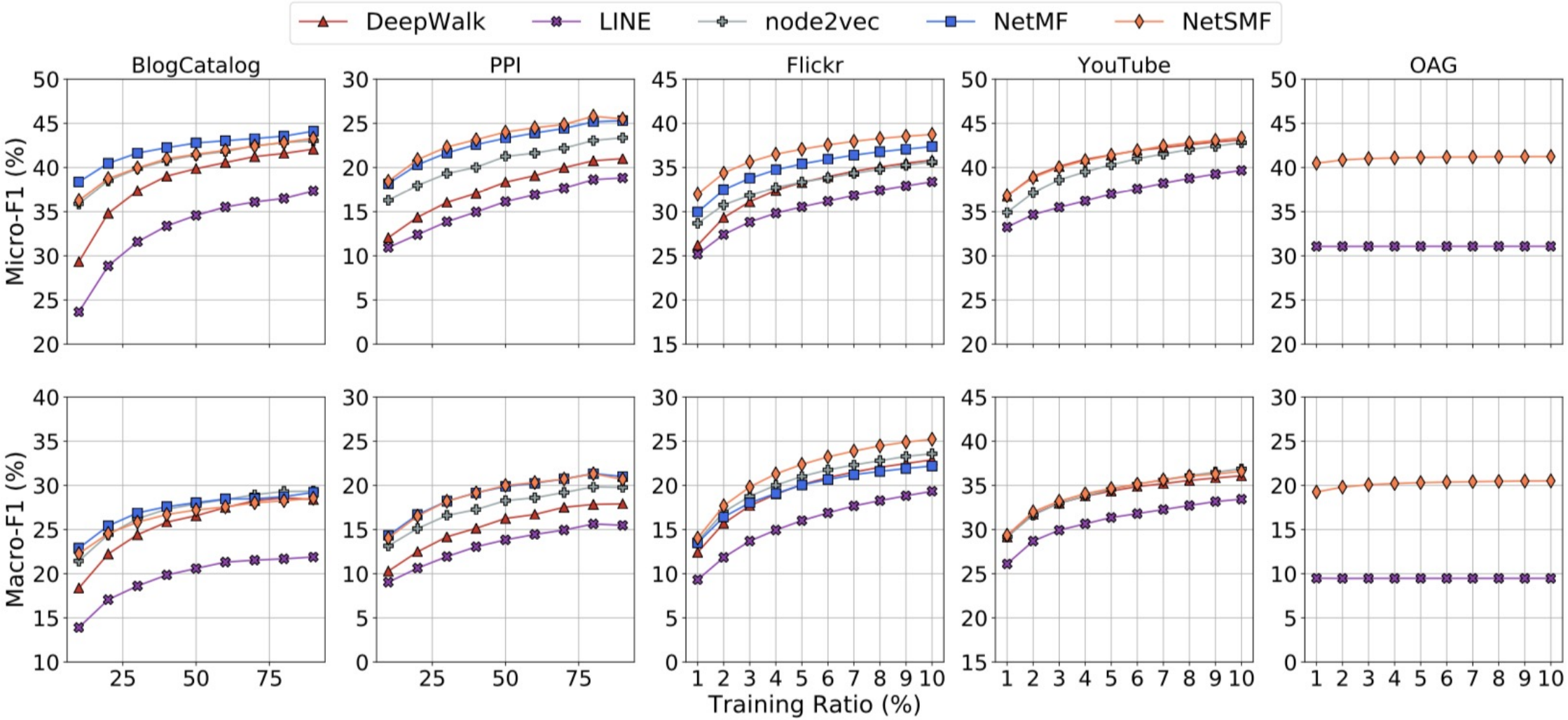
- Construct a random walk matrix polynomial sparsifier, $\widetilde{\boldsymbol{L}}$
- Construct a NetMF matrix sparsifier.

$$\text{trunc\_log}^{\circ}\left(\frac{\text{vol}(G)}{b}\boldsymbol{D}^{-1}(\boldsymbol{D}-\widetilde{\boldsymbol{L}})\boldsymbol{D}^{-1}\right)$$

- Factorize the constructed matrix

1. J. Qiu, Y. Dong, H. Ma, J. Li, C. Wang, K. Wang, and J. Tang. NetSMF: Large-Scale Network Embedding as Sparse Matrix Factorization. WWW'19.

Results



** Code available at https://github.com/xptree/NetSMF

NE as Sparse Matrix Factorization

- node-context set $\mathcal{D} = E$ (sparsity)
- To avoid the trivial solution $(r_i = c_j, r_i^T c_j \to \infty, s.t. \hat{p} \to 1)$
- Local negative samples drawn from

$$P_{\mathcal{D},j} \propto \sum_{i:(i,j)\in\mathcal{D}} p_{i,j}$$

- Modify the loss (sum over the edge-->sparse)

$$l = -\sum_{(i,j)\in\mathcal{D}} [p_{i,j} \ln \sigma(r_i^T c_j) + \lambda P_{\mathcal{D},j} \ln \sigma(-r_i^T c_j)]$$

NE as Sparse Matrix Factorization

- Let the partial derivative w.r.t. $r_i^T c_j$ be zero

$$r_i^T c_j = \ln p_{i,j} - \ln(\lambda P_{D,j}), \quad (v_i, v_j) \in \mathcal{D}$$

- Matrix to be factorized (sparse)

$$M_{i,j} = \begin{cases} \ln p_{i,j} - \ln(\lambda P_{D,j}) & , (v_i, v_j) \in \mathcal{D} \\ 0 & , (v_i, v_j) \notin \mathcal{D} \end{cases}$$

NE as Sparse Matrix Factorization

- Compared with matrix factorization method (e.g., NetMF)

$$\log\left(\frac{\mathrm{vol}(G)}{b}\left(\frac{1}{T}\sum_{r=1}^{T}\left(\boldsymbol{D}^{-1}\boldsymbol{A}\right)^{r}\right)\boldsymbol{D}^{-1}\right) \quad \textbf{v.s.} \quad M_{i,j}=\begin{cases}\ln p_{i,j}-\ln(\lambda P_{D,j}) & ,(v_i,v_j)\in\mathcal{D}\\ 0 & ,(v_i,v_j)\notin\mathcal{D}\end{cases}$$

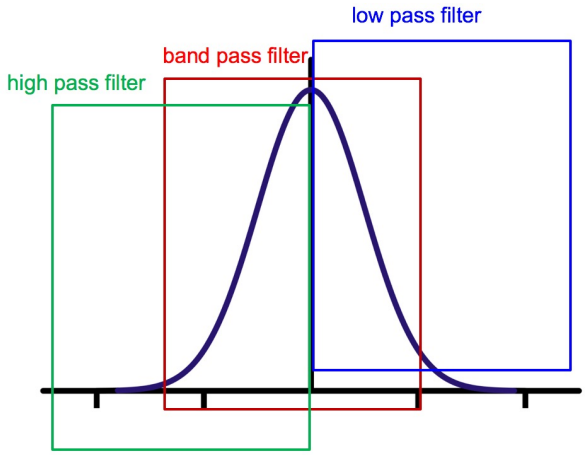
- Sparsity (local structure and local negative samples)→ much faster and scalable (e.g., randomized tSVD, O(|*E*|))
- The optimization (single thread) is much faster than SGD used in DeepWalk, LINE, etc. and is still scalable!!!
- Challenge: may lose high order information!
- Improvement via spectral propagation

NE Enhancement via Spectral Propagation

- the form of the spectral filter

$$\widetilde{L} = U diag([g(\lambda_1), ..., g(\lambda_n)])U^T$$

$$g(\lambda) = e^{-\frac{1}{2}[(\lambda-\mu)^2-1]\theta}$$

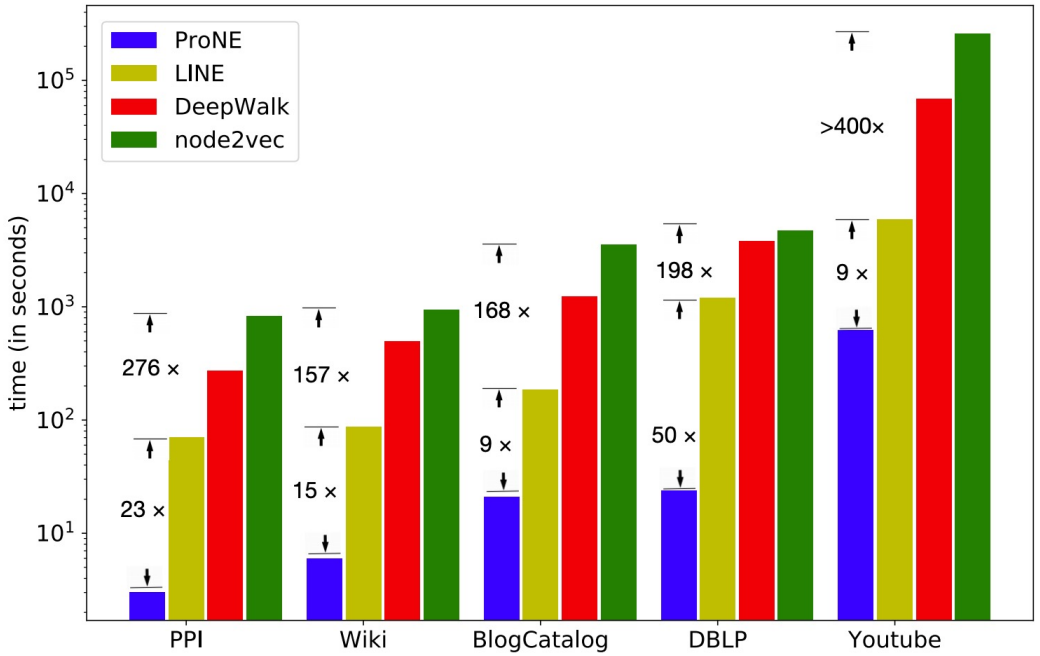


- Band-pass (low-pass, high-pass)
- pass eigenvalues within a certain range and weaken eigenvalues outside that range
- amplify local and global network information

Complexity of ProNE

- Spectral propagation only involves sparse matrix multiplication! The complexity is linear!
- Sparse matrix factorization + spectral propagation = $O(|V|d^2 + k|E|)$

Results



* ProNE (1 thread) v.s. Others (20 threads)

* 10 minutes on Youtube (~1M nodes)

Dataset	*DeepWalk*	*LINE*	*node2vec*	*ProNE*
PPI	272	70	828	**3**
Wiki	494	87	939	**6**
BlogCatalog	1,231	185	3,533	**21**
DBLP	3,825	1,204	4,749	**24**
Youtube	68,272	5,890	>5days	**627**

** Code available at https://github.com/THUDM/ProNE

Effectiveness experiments

Dataset	training ratio	0.1	0.3	0.5	0.7	0.9
PPI	DeepWalk	16.4	19.4	21.1	22.3	22.7
	LINE	16.3	20.1	21.5	22.7	23.1
	node2vec	16.2	19.7	21.6	23.1	24.1
	GraRep	15.4	18.9	20.2	20.4	20.9
	HOPE	16.4	19.8	21.0	21.7	22.5
	ProNE (SMF)	15.8	20.6	22.7	23.7	24.2
	ProNE	**18.2**	**22.7**	**24.6**	**25.4**	**25.9**
	($\pm\sigma$)	($\pm$0.5)	($\pm$0.3)	($\pm$0.7)	($\pm$1.0)	($\pm$1.1)
Wiki	DeepWalk	40.4	45.9	48.5	49.1	49.4
	LINE	**47.8**	50.4	51.2	51.6	52.4
	node2vec	45.6	47.0	48.2	49.6	50.0
	GraRep	47.2	49.7	50.6	50.9	51.8
	HOPE	38.5	39.8	40.1	40.1	40.1
	ProNE (SMF)	47.6	51.6	53.2	53.5	53.9
	ProNE	47.3	**53.1**	**54.7**	**55.2**	**57.2**
	($\pm\sigma$)	($\pm$0.7)	($\pm$0.4)	($\pm$0.8)	($\pm$0.8)	($\pm$1.3)
BlogCatalog	DeepWalk	36.2	39.6	40.9	41.4	42.2
	LINE	28.2	30.6	33.2	35.5	36.8

Dataset	training ratio	0.01	0.03	0.05	0.07	0.09
DBLP	DeepWalk	49.3	55.0	57.1	57.9	58.4
	LINE	48.7	52.6	53.5	54.1	54.5
	node2vec	48.9	55.1	57.0	58.0	58.4
	GraRep	50.5	52.6	53.2	53.5	53.8
	HOPE	**52.2**	55.0	55.9	56.3	56.6
	ProNE (SMF)	50.8	54.9	56.1	56.7	57.0
	ProNE	48.8	**56.2**	**58.0**	**58.8**	**59.2**
	($\pm\sigma$)	($\pm$1.0)	($\pm$0.5)	($\pm$0.2)	($\pm$0.2)	($\pm$0.1)
Youtube	DeepWalk	38.0	40.1	41.3	42.1	42.8
	LINE	33.2	35.5	37.0	38.2	39.3
	ProNE (SMF)	36.5	40.2	41.2	41.7	42.1
	ProNE	**38.2**	**41.4**	**42.3**	**42.9**	**43.3**
	($\pm\sigma$)	($\pm$0.8)	($\pm$0.3)	($\pm$0.2)	($\pm$0.2)	($\pm$0.2)

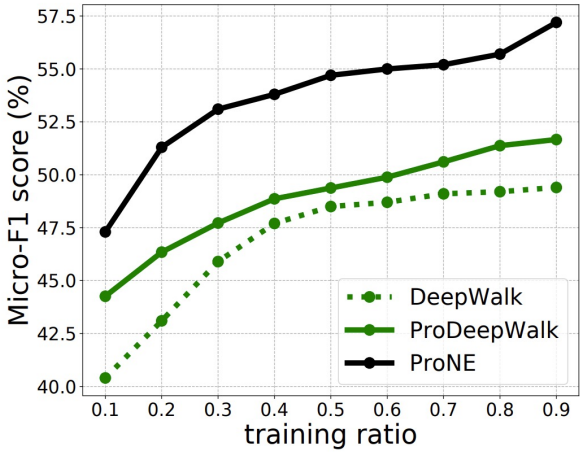
* ProNE (SMF) = ProNE w/

n

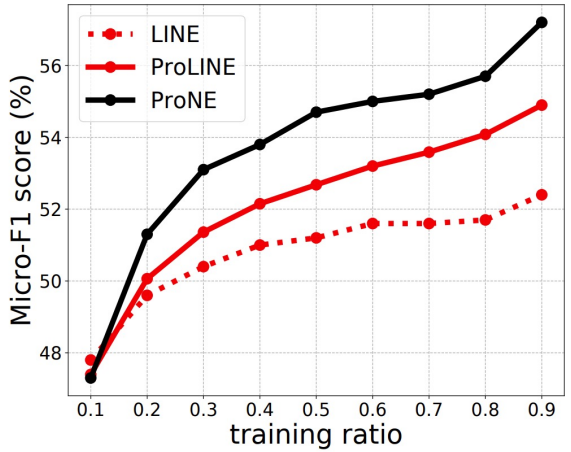
Embed 100,000,000 nodes by one thread:
29 hours with **performance superiority**

** Code available at https://github.com/THUDM/ProNE

Spectral Propagation: *k*-way Cheeger

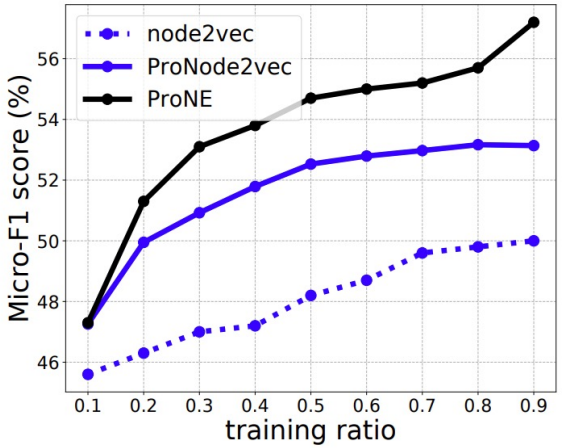


(a) ProDeepWalk

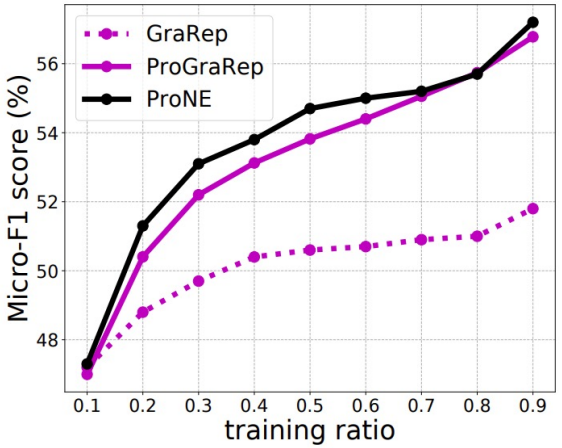


(b) ProLINE

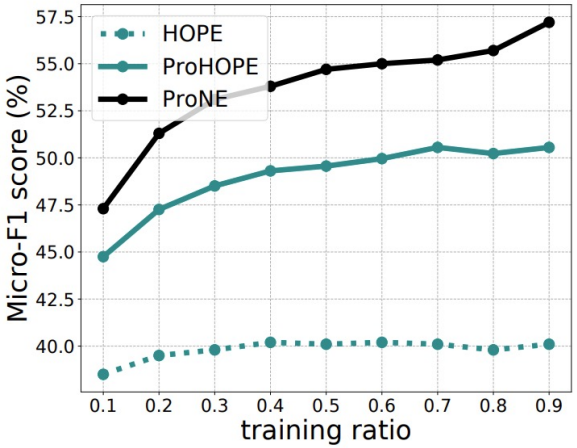
$$\frac{\lambda_k}{2} \leq \rho_G(k) \leq O(k^2)\sqrt{\lambda_k}$$



(c) ProNode2vec

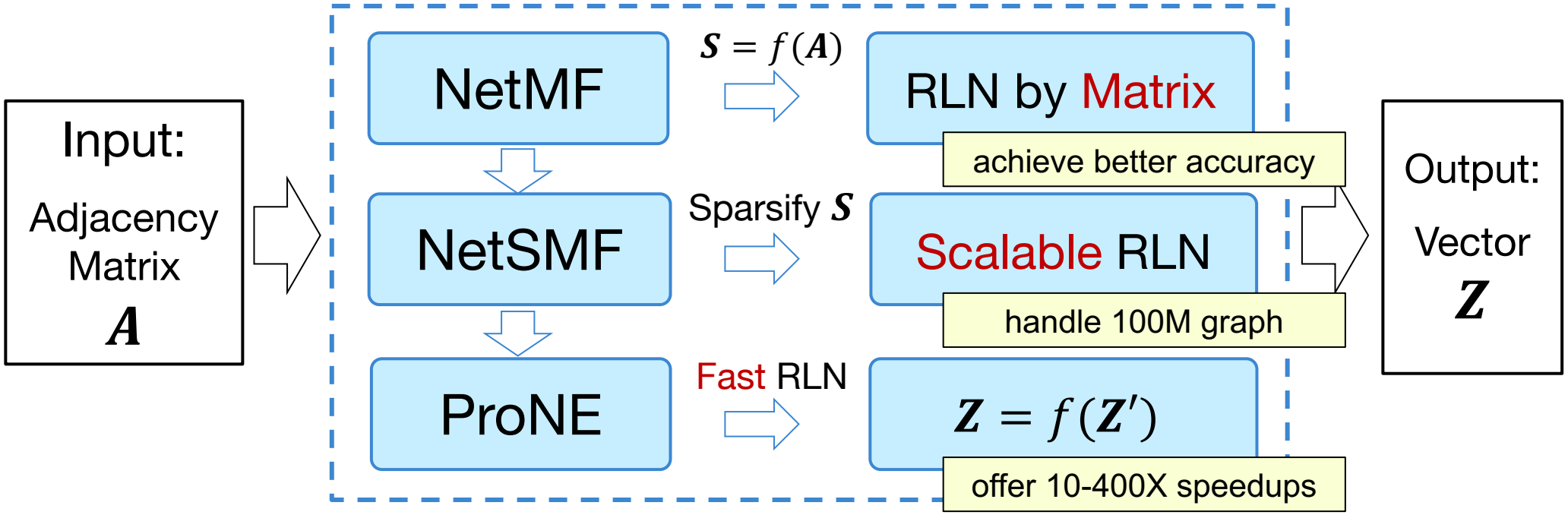


(d) ProGraRep



(e) ProHOPE

Representation Learning on Networks



1. Qiu et al. Network embedding as matrix factorization: unifying deepwalk, line, pte, and node2vec. *WSDM'18.* **The most cited paper in WSDM'18 as of May 2019**
2. J. Qiu, Y. Dong, H. Ma, J. Li, C. Wang, K. Wang, and J. Tang. NetSMF: Large-Scale Network Embedding as Sparse Matrix Factorization. WWW'19.
3. J. Zhang, Y. Dong, Y. Wang, J. Tang, and M. Ding. ProNE: Fast and Scalable Network Representation Learning. IJCAI'19.

GRL: NE&GNN

Network Embedding ⇔ Matrix Factorization

Graph Neural Networks ⇨ GNN Pre-Training

Graph Data & Benchmarks

OPEN GRAPH BENCHMARK

https://alchemy.tencent.com/

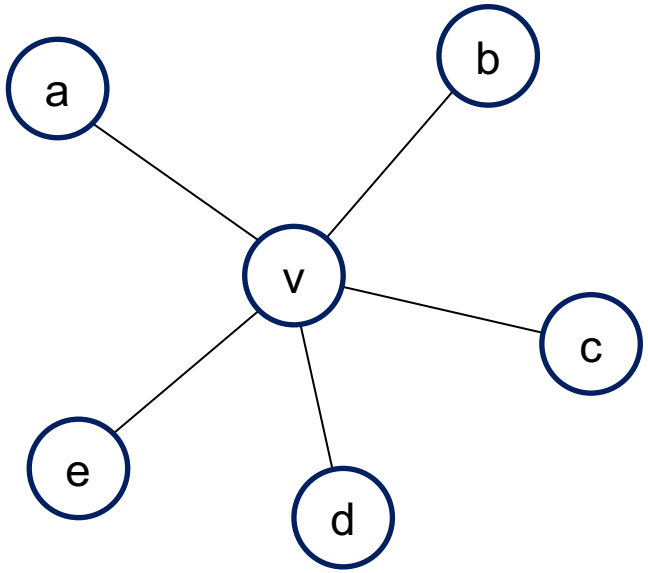
Microsoft Academic

AMiner

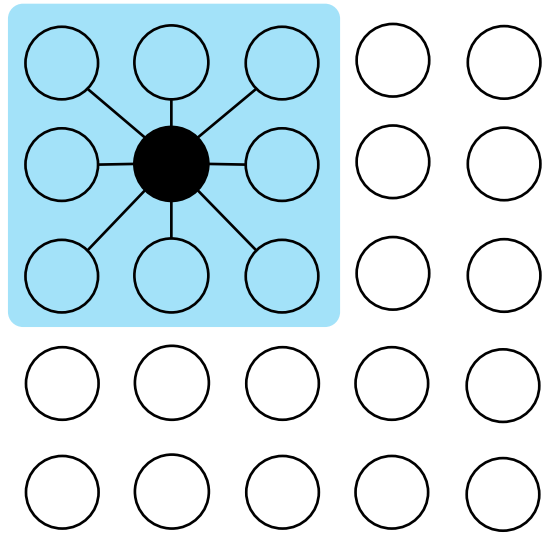
CogDL

https://github.com/thudm/cogdl

Graph Neural Networks



1. Choose neighborhood
2. Determine the order of selected neighbors
3. Parameter sharing



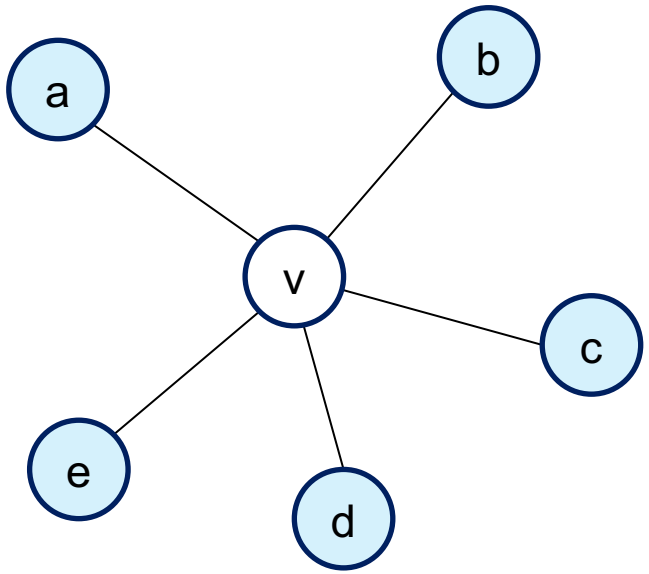
Graph Convolution **CNN**

Neighborhood Aggregation:

- Aggregate neighbor information and pass into a neural network
- It can be viewed as a center-surround filter in CNN---graph convolutions!

1. Niepert et al. Learning Convolutional Neural Networks for Graphs. In ICML 2016
2. Defferrard et al. Convolutional Neural Networks on Graphs with Fast Locailzied Spectral Filtering. In NIPS 2016
3. Huawei Shen. Graph Neural Networks. A video talk (in Chinese), June 2020. https://dl.ccf.org.cn/audioVideo/detail.html?id=4966039790962688&_ack=1

Graph Convolutional Networks



Non-linear activation function (e.g., ReLU)

parameters in layer k

$$\boldsymbol{h}_v^k = \sigma\left(\boldsymbol{W}^k \sum_{u \in \mathrm{N}(v) \cup v} \frac{\boldsymbol{h}_u^{k-1}}{\sqrt{|N(u)||N(v)|}}\right)$$

node v's embedding at layer k

the neighbors of node v

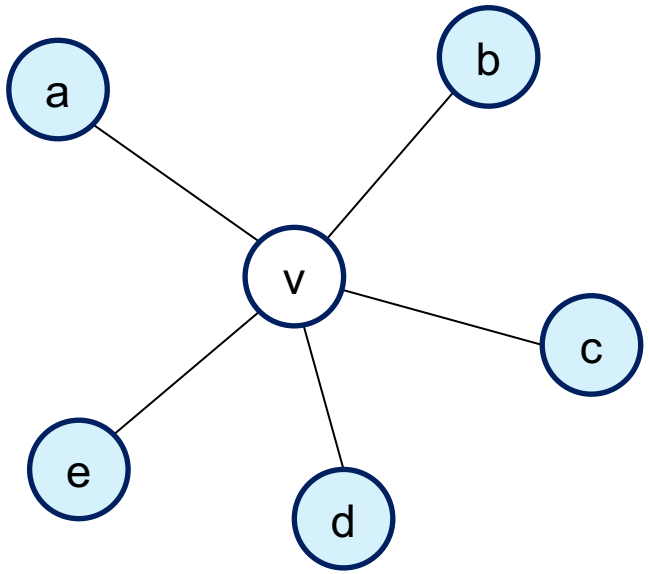
$$\boldsymbol{H}^k = \sigma\left(\widehat{\boldsymbol{A}}\boldsymbol{H}^{(k-1)}\boldsymbol{W}^{(k)}\right)$$

normalized Laplacian matrix

Aggregate info from neighborhood via the normalized Laplacian matrix

1. Kipf et al. Semisupervised Classification with Graph Convolutional Networks. ICLR 2017

Graph Convolutional Networks



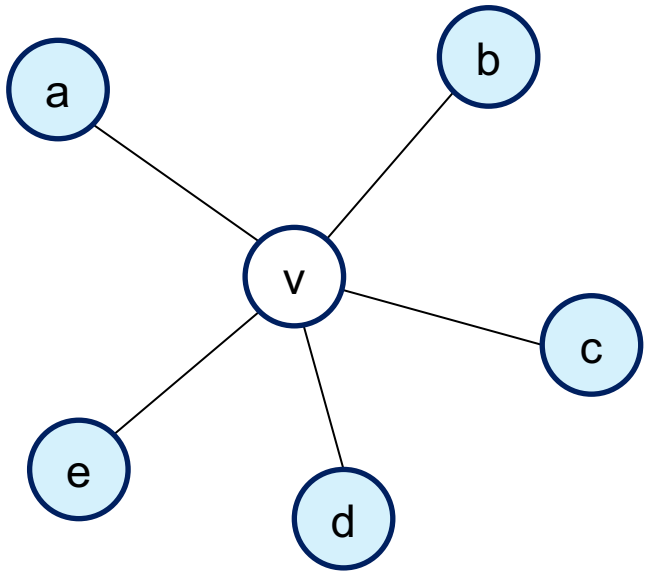
Aggregate from v's neighbors

$$\boldsymbol{h}_v^k = \sigma(\boldsymbol{W}^k \sum_{u \in \mathrm{N}(v)} \frac{\boldsymbol{h}_u^{k-1}}{\sqrt{|N(u)||N(v)|}} + \boldsymbol{W}^k \sum_{v} \frac{\boldsymbol{h}_v^{k-1}}{\sqrt{|N(v)||N(v)|}})$$

Aggregate from itself

1. Kipf et al. Semisupervised Classification with Graph Convolutional Networks. ICLR 2017

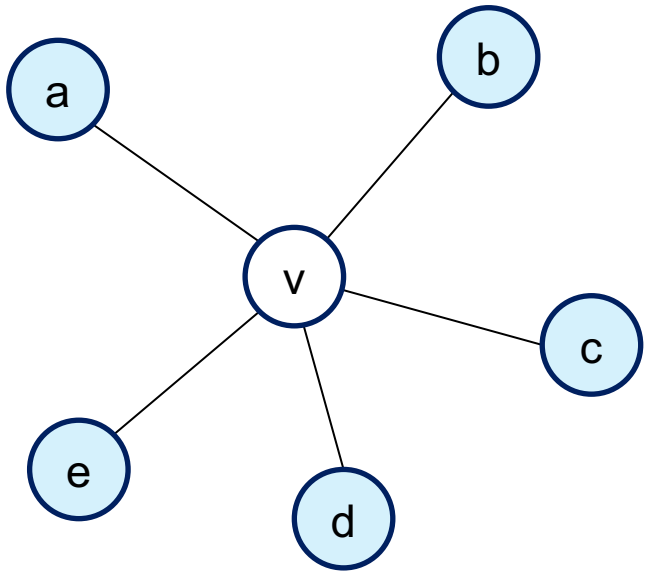
Graph Convolutional Networks



The same parameters for both its neighbors & itself

$$\boldsymbol{h}_v^k = \sigma\left(\boldsymbol{W}^k \sum_{u \in \mathrm{N}(v)} \frac{\boldsymbol{h}_u^{k-1}}{\sqrt{|N(u)||N(v)|}} + \boldsymbol{W}^k \sum_{v} \frac{\boldsymbol{h}_v^{k-1}}{\sqrt{|N(v)||N(v)|}}\right)$$

Graph Convolutional Networks

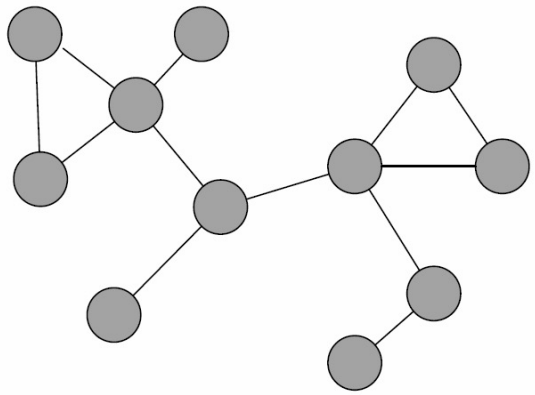


$$\boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{A}\boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{H}^{(k-1)}\boldsymbol{W}^{(k)}$$

$$\boldsymbol{h}_v^k = \sigma(\boldsymbol{W}^k \sum_{u \in \mathrm{N}(v)} \frac{\boldsymbol{h}_u^{k-1}}{\sqrt{|N(u)||N(v)|}} + \boldsymbol{W}^k \sum_{v} \frac{\boldsymbol{h}_v^{k-1}}{\sqrt{|N(v)||N(v)|}})$$

$$\boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{I}\boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{H}^{(k-1)}\boldsymbol{W}^{(k)}$$

Graph Convolutional Networks



$G = (V, E, \boldsymbol{A})$

$\boldsymbol{H}^0 = \boldsymbol{X}$

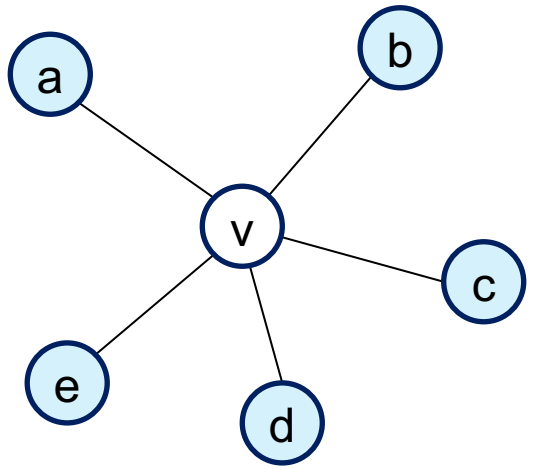
Input

$$\boldsymbol{H}^k = \sigma\left(\boldsymbol{D}^{-\frac{1}{2}}(\boldsymbol{A}+\boldsymbol{I})\boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{H}^{(k-1)}\boldsymbol{W}^{(k)}\right)$$

$$\Rightarrow \boldsymbol{Z} = \boldsymbol{H}^K$$

- Model training
  - The common setting is to have an end to end training framework with a supervised task
- Benefits: Parameter sharing for all nodes
  - #parameters is subline in |V|
  - Enable inductive learning for new nodes

GraphSAGE



GCN

$$\boldsymbol{h}_v^k = \sigma(\boldsymbol{W}^k \sum_{u \in \mathrm{N}(v) \cup v} \frac{\boldsymbol{h}_u^{k-1}}{\sqrt{|N(u)||N(v)|}})$$

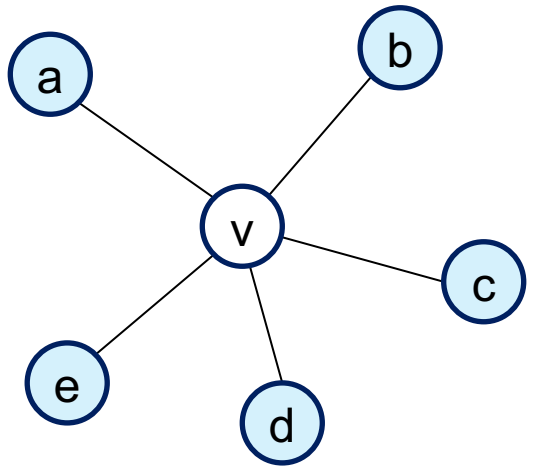
GraphSage

$$\boldsymbol{h}_v^k = \sigma([\boldsymbol{A}^k \cdot \mathrm{AGG}(\{\boldsymbol{h}_u^{k-1}, \forall u \in N(v)\}), \boldsymbol{B}^k \boldsymbol{h}_v^{k-1}])$$

Generalized aggregation: any differentiable function that maps set of vectors to a single vector

1. Hamilton et al. Inductive Representation Learning on Large Graphs. NIPS 2017
2. Hamiltion & Tang, AAAI 2019 Tutorial on Graph Representation Learning

GraphSAGE



GCN

$$\boldsymbol{h}_v^k = \sigma\left(\boldsymbol{W}^k \sum_{u \in \mathrm{N}(v) \cup v} \frac{\boldsymbol{h}_u^{k-1}}{\sqrt{|N(u)||N(v)|}}\right)$$

GraphSage

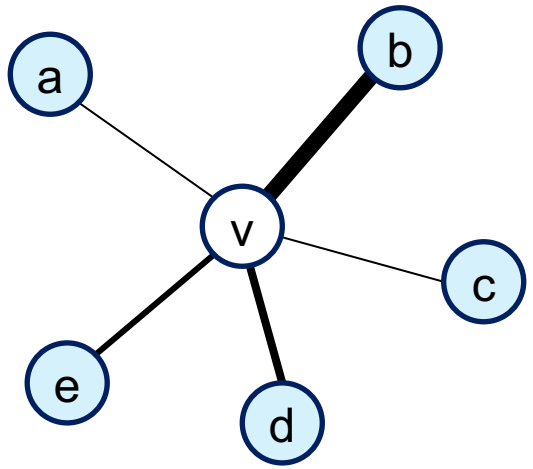
Instead of summation, it concatenates neighbor & self embeddings

$$\boldsymbol{h}_v^k = \sigma\left([\boldsymbol{A}^k \cdot \mathrm{AGG}\left(\{\boldsymbol{h}_u^{k-1}, \forall u \in N(v)\}\right), \boldsymbol{B}^k \boldsymbol{h}_v^{k-1}]\right)$$

Generalized aggregation: any differentiable function that maps set of vectors to a single vector

1. Hamilton et al. Inductive Representation Learning on Large Graphs. NIPS 2017
2. Hamiltion & Tang, AAAI 2019 Tutorial on Graph Representation Learning

GNN: Graph Attention



GCN

$$\boldsymbol{h}_v^k = \sigma(\boldsymbol{W}^k \sum_{u \in \mathrm{N}(v) \cup v} \frac{\boldsymbol{h}_u^{k-1}}{\sqrt{|N(u)||N(v)|}})$$

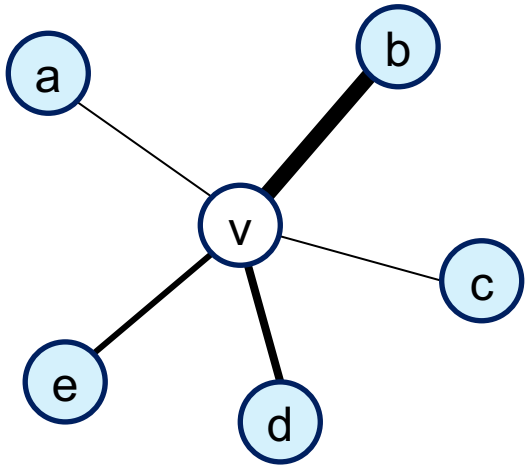
Graph Attention

$$\boldsymbol{h}_v^k = \sigma(\sum_{u \in \mathrm{N}(v) \cup v} \alpha_{v,u} \boldsymbol{W}^k \boldsymbol{h}_u^{k-1})$$

Learned attention weights

1. Velickovic et al. Graph Attention Networks. ICLR 2018

GNN: Graph Attention



$$\alpha_{v,u} = \frac{\exp\left(\text{LeakyReLU}\left(\mathbf{a}^\top [\mathbf{Q}\mathbf{h}_v, \mathbf{Q}\mathbf{h}_u]\right)\right)}{\sum_{u' \in N(v) \cup \{v\}} \exp\left(\text{LeakyReLU}\left(\mathbf{a}^\top [\mathbf{Q}\mathbf{h}_v, \mathbf{Q}\mathbf{h}_{u'}]\right)\right)}$$

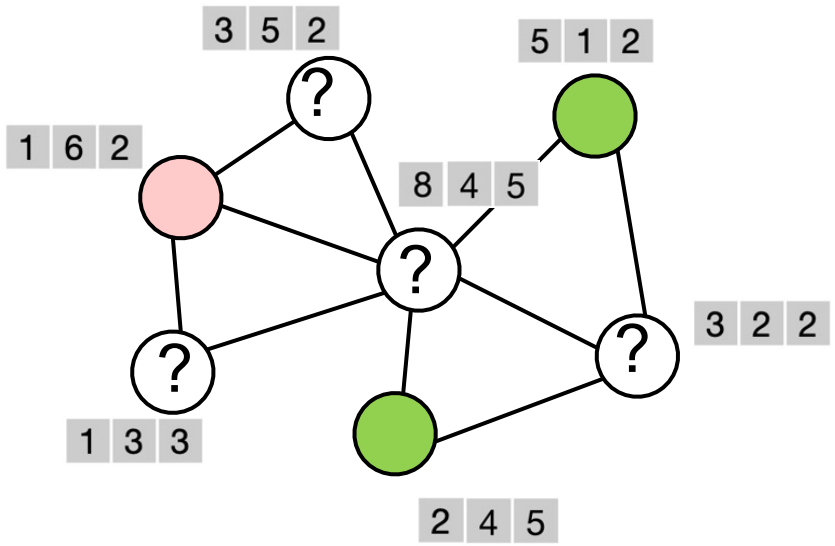
Various ways to define attention!

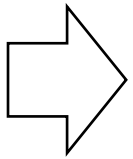
1. Velickovic et al. Graph Attention Networks. ICLR 2018

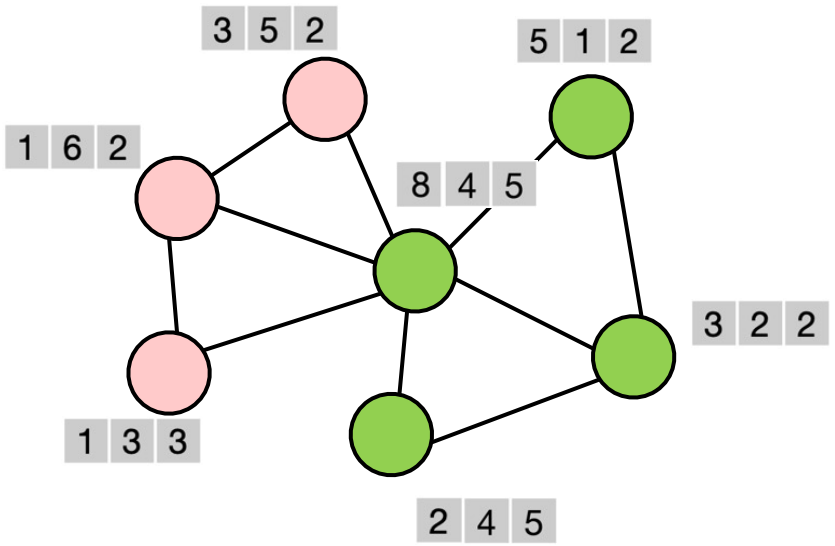
Revisiting Semi-Supervised Learning on Graphs

- W. Feng, J. Zhang, Y. Dong, Y. Han, H. Luan, Q. Xu, Q. Yang, E. Kharlamov, and J. Tang. Graph Random Neural Networks for Semi-Supervised Learning on Graphs. NeurIPS'20. https://arxiv.org/abs/2005.11079
- Code & data for Grand: https://github.com/Grand20/grand

Semi-Supervised Learning on Graphs







Input: a partially labeled & attributed graph

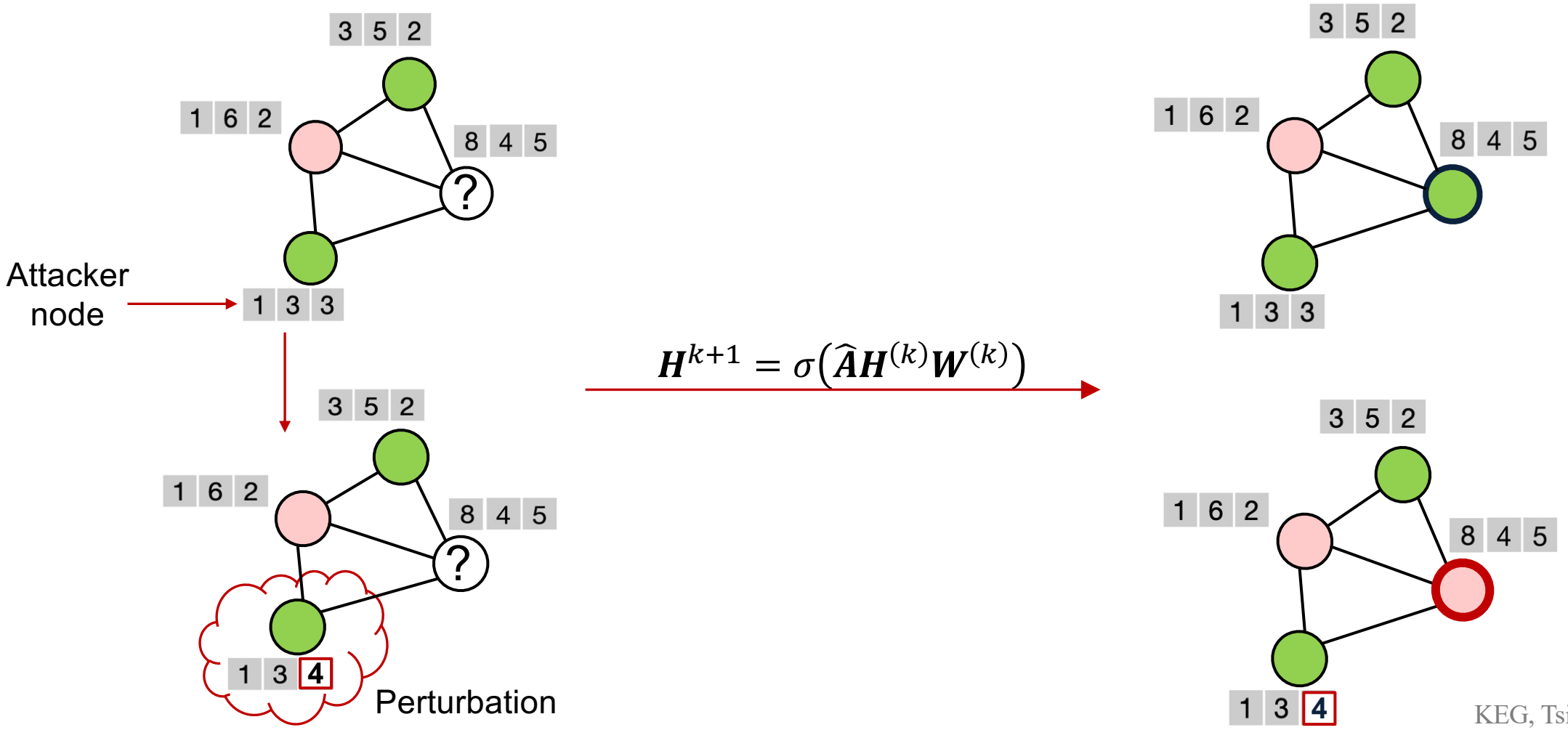
Output: infer the labels of unlabeled nodes

Graph Neural Networks

$$\boldsymbol{H}^{k+1} = \sigma\left(\widehat{\boldsymbol{A}}\boldsymbol{H}^{(k)}\boldsymbol{W}^{(k)}\right)$$

a deterministic propagation

1. Each node is highly dependent with its neighbors, making GNNs **non-robust** to noises



Graph Neural Networks

$$\boldsymbol{H}^{k+1} = \sigma\left(\widehat{\boldsymbol{A}}\boldsymbol{H}^{(k)}\boldsymbol{W}^{(k)}\right)$$

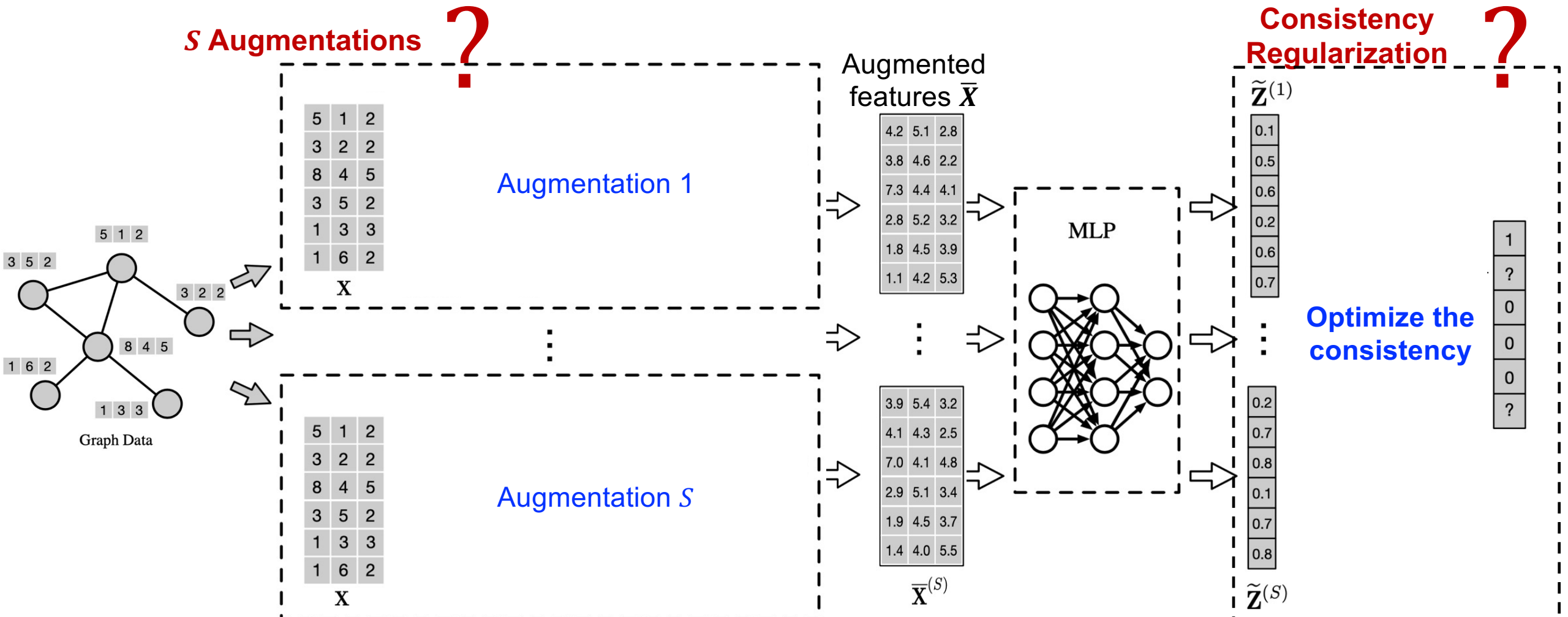
feature **propagation**
is coupled with
non-linear transformation

1. Each node is highly dependent with its neighbors, making GNNs **non-robust** to noises

2. Stacking many GNN layers may cause **over-fitting** & **over-smoothing.**

Graph Random Neural Networks (GRAND)

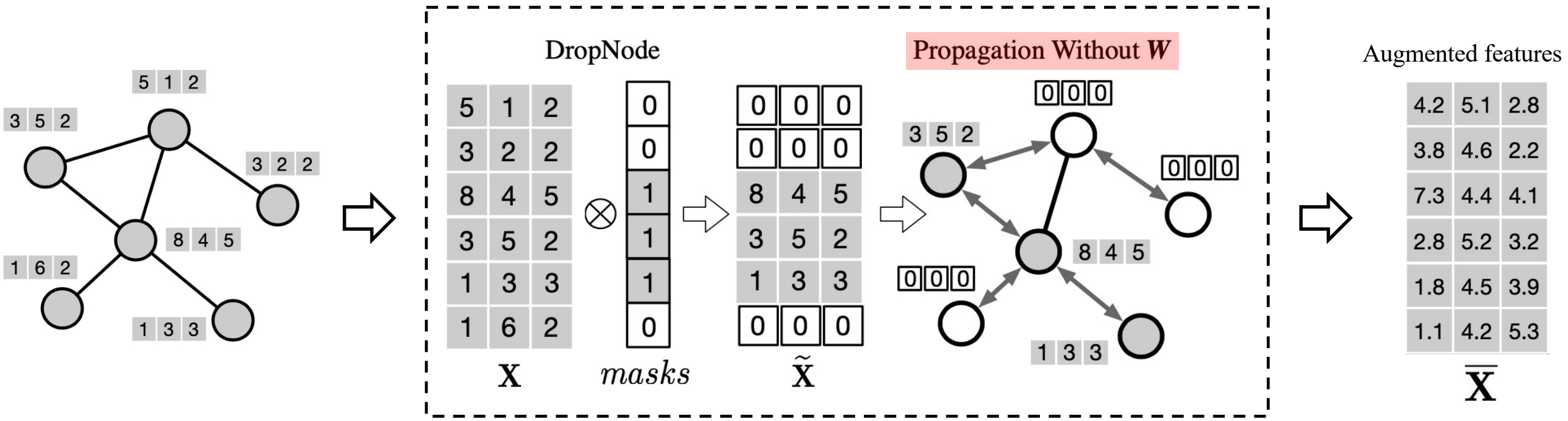
- Consistency Regularized Training:
  - Generates S data augmentations of the graph
  - Optimizing the consistency among S augmentations of the graph.



GRAND: Random Propagation for Graph Data Augmentation

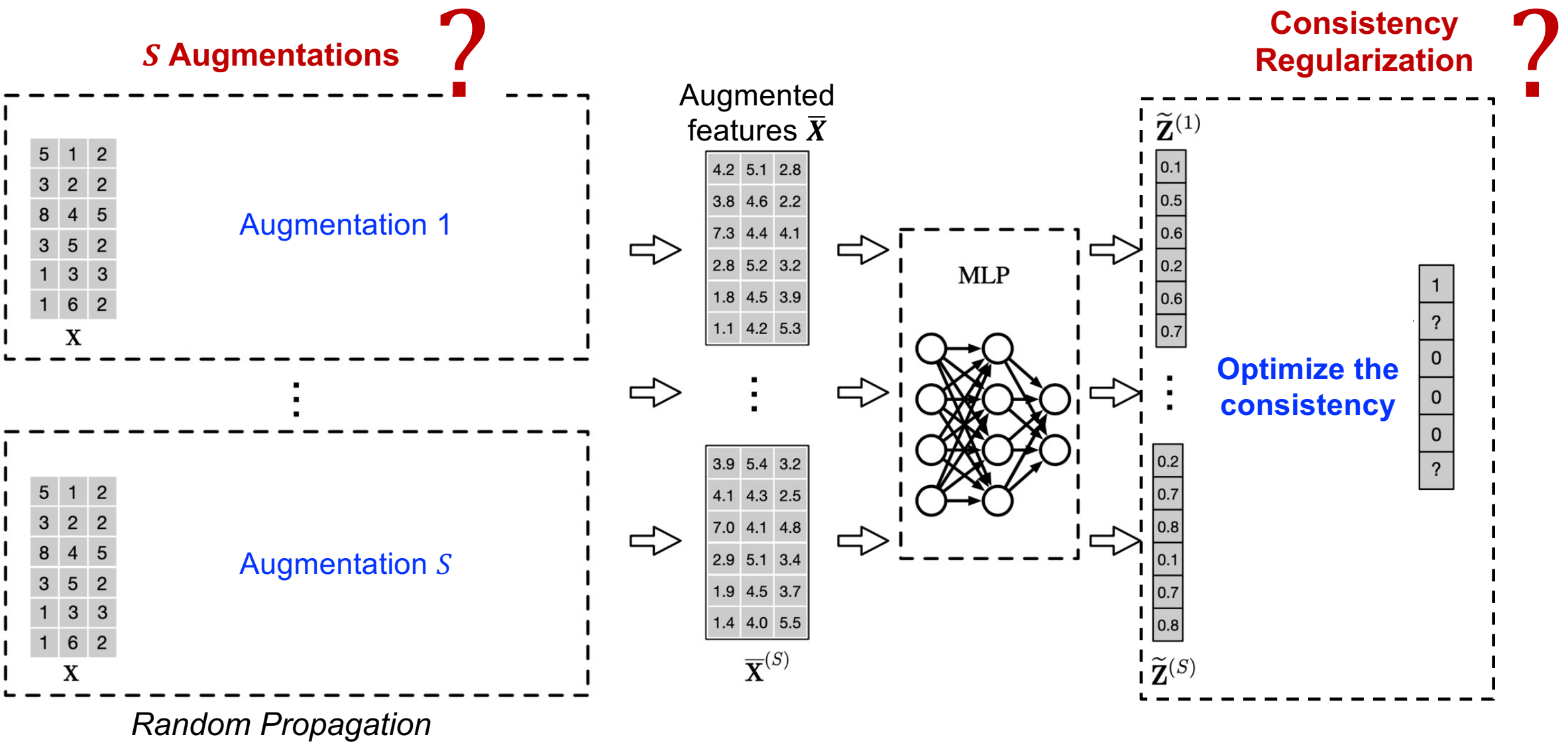
- **Random Propagation** (DropNode + Propagation):
  - Each node is enabled to be not sensitive to specific neighborhoods.
  - Decouple feature propagation from feature transformation.

Random Propagation for Augmentation



$$\overline{\mathbf{X}} = \overline{\mathbf{A}}\tilde{\mathbf{X}}$$

Graph Random Neural Networks (GRAND)



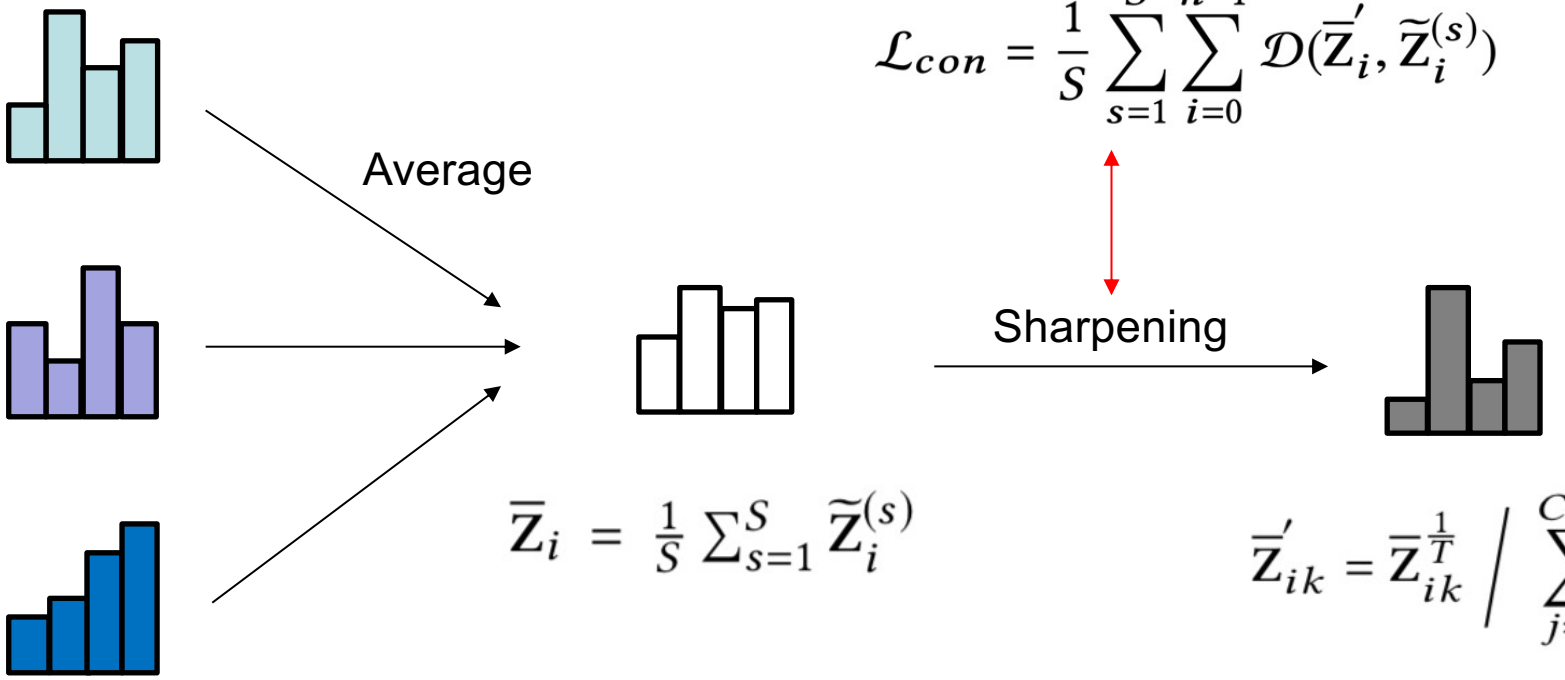
GRAND: Consistency Regularization

$$\mathcal{L}_{sup} = -\frac{1}{S}\sum_{s=1}^{S}\sum_{i=0}^{m-1}\mathbf{Y}_i^{\top}\log\widetilde{\mathbf{Z}}_i^{(s)}$$

$$\mathcal{L}_{con} = \frac{1}{S}\sum_{s=1}^{S}\sum_{i=0}^{n-1}\mathcal{D}(\overline{\mathbf{Z}}_i', \widetilde{\mathbf{Z}}_i^{(s)})$$

$$\mathcal{L} = \mathcal{L}_{sup} + \lambda\mathcal{L}_{con}$$

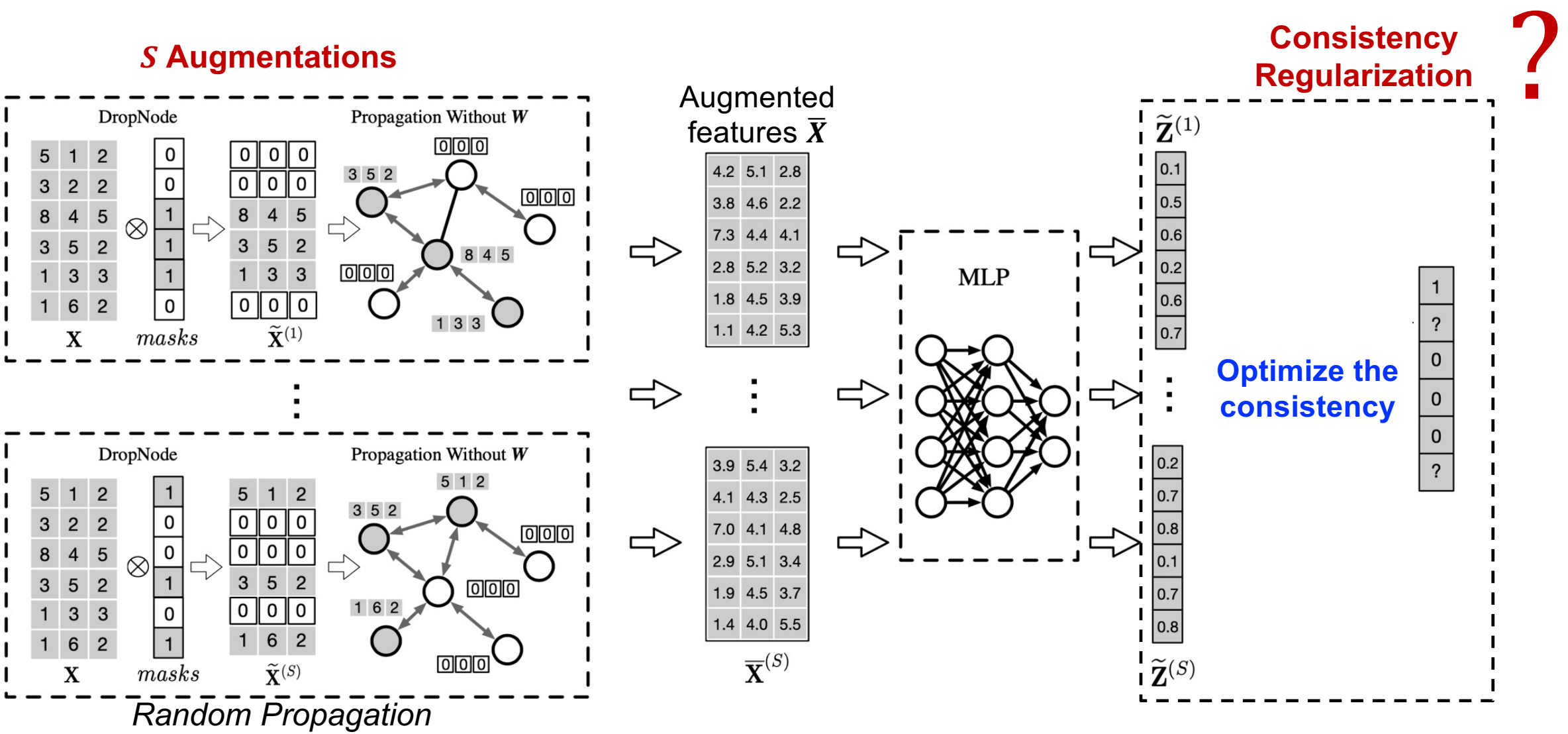
Distributions of a node after augmentations



$$\overline{\mathbf{Z}}_i = \frac{1}{S}\sum_{s=1}^{S}\widetilde{\mathbf{Z}}_i^{(s)}$$

$$\overline{\mathbf{Z}}'_{ik} = \overline{\mathbf{Z}}_{ik}^{\frac{1}{T}} \Big/ \sum_{j=0}^{C-1}\overline{\mathbf{Z}}_{ij}^{\frac{1}{T}}$$

Graph Random Neural Networks (GRAND)



Consistency Regularized Training Algorithm

Input:

Adjacency matrix $\hat{\mathbf{A}}$, feature matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$, times of augmentations in each epoch S, DropNode probability δ.

Output:

Prediction $\mathbf{Z}$.

while not convergence **do**
  for $s = 1 : S$ **do**
    Apply DropNode via Algorithm 1: $\widetilde{\mathbf{X}}^{(s)} \sim \text{DropNode}(\mathbf{X}, \delta)$.
    Perform propagation: $\overline{\mathbf{X}}^{(s)} = \frac{1}{K+1} \sum_{k=0}^{K} \hat{\mathbf{A}}^k \widetilde{\mathbf{X}}^{(s)}$.
    Predict class distribution using MLP: $\widetilde{\mathbf{Z}}^{(s)} = P(\mathbf{Y} | \overline{\mathbf{X}}^{(s)}; \Theta)$.
  end for
  Compute supervised classification loss $\mathcal{L}_{sup}$ via Eq. 4 and consistency regularization loss via Eq. 6.
  Update the parameters Θ by gradients descending:

$$\nabla_{\Theta} \mathcal{L}_{sup} + \lambda \mathcal{L}_{con}$$

end while
Output prediction $\mathbf{Z}$ via Eq. 8.

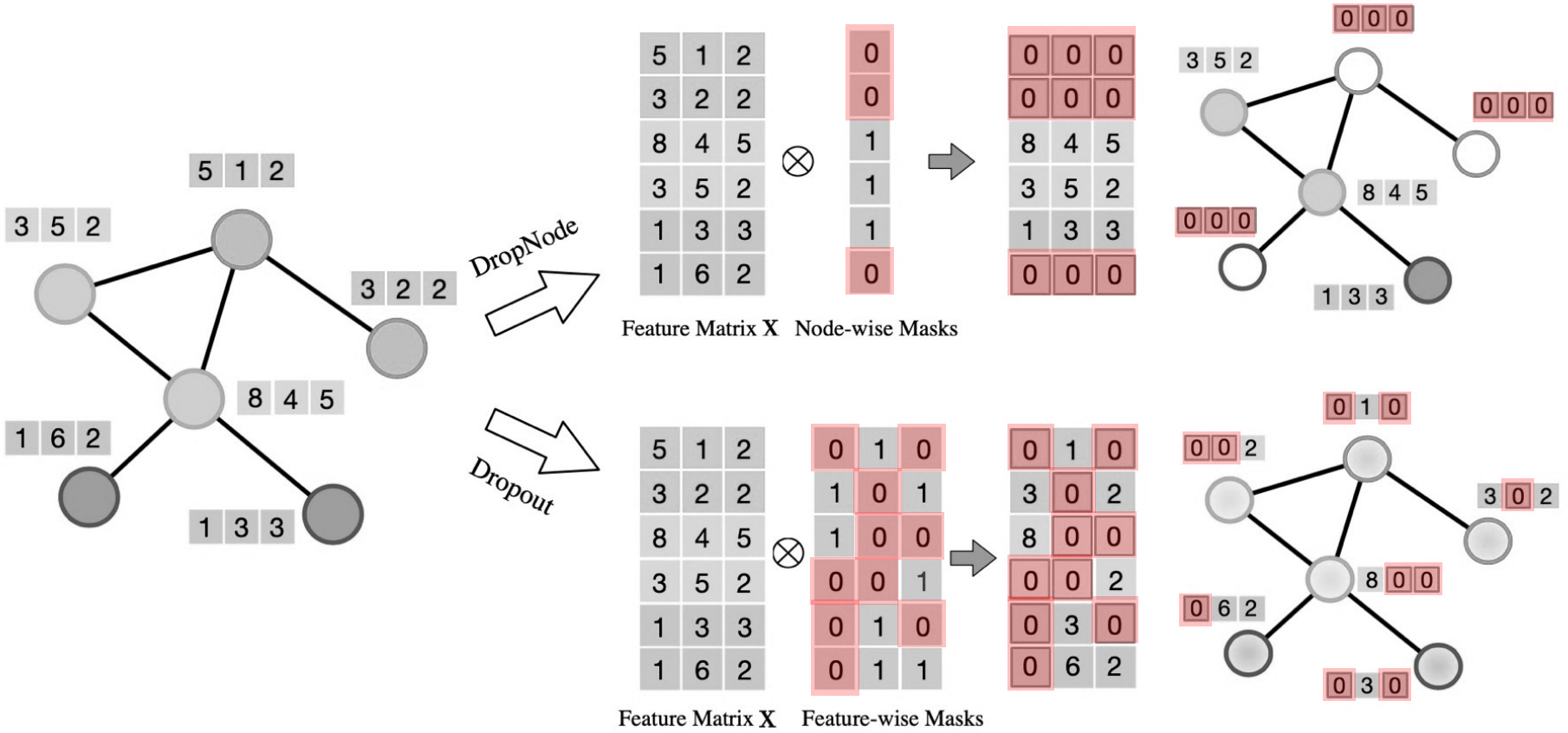
Generate S Augmentations

Consistency Regularization

- W. Feng, J. Zhang, Y. Dong, Y. Han, H. Luan, Q. Xu, Q. Yang, E. Kharlamov, and J. Tang. Graph Random Neural Networks for Semi-Supervised Learning on Graphs. NeurIPS'20. https://arxiv.org/abs/2005.11079
- Code & data for Grand: https://github.com/Grand20/grand

GRAND: DropNode vs Dropout

- Dropout drops each element in $\boldsymbol{X}$ independently
- DropNode drops the entire features of selected nodes, i.e., the row vectors of $\boldsymbol{X}$, randomly
  - Theoretically, Dropout is an adaptive L_2 regularization.



Graph Random Neural Networks (GRAND)

- With Consistency Regularization Loss:
  - Random propagation can enforce the consistency of the classification confidence between each node and its all multi-hop neighborhoods.

$$\mathbb{E}_\epsilon\left(\mathcal{L}_{con}\right) \approx \mathcal{R}^c(\mathbf{W}) = \sum_{i=0}^{n-1} z_i^2(1-z_i)^2 \mathrm{Var}_\epsilon\left(\overline{\mathbf{A}}_i \widetilde{\mathbf{X}} \cdot \mathbf{W}\right)$$

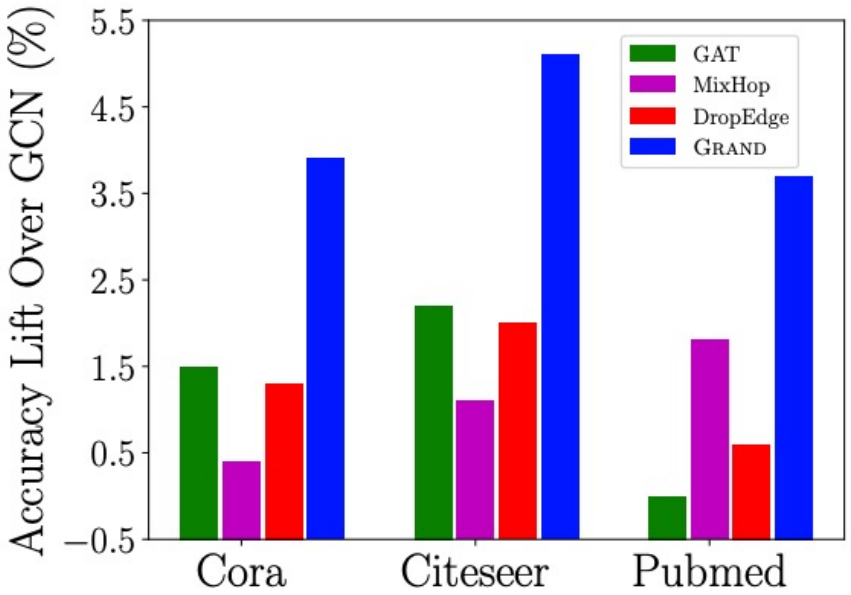
$$\mathcal{R}^c_{DN}(\mathbf{W}) = \frac{\delta}{1-\delta} \sum_{j=0}^{n-1} \left[(\mathbf{X}_j \cdot \mathbf{W})^2 \sum_{i=0}^{n-1} (\overline{\mathbf{A}}_{ij})^2 z_i^2 (1-z_i)^2 \right]$$

$$\mathcal{R}^c_{Do}(\mathbf{W}) = \frac{\delta}{1-\delta} \sum_{h=0}^{d-1} \mathbf{W}_h^2 \sum_{j=0}^{n-1} \left[\mathbf{X}_{jh}^2 \sum_{i=0}^{n-1} z_i^2 (1-z_i)^2 (\overline{\mathbf{A}}_{ij})^2 \right]$$

- With Supervised Cross-Entropy Loss:
  - Random propagation can enforce the consistency of the classification confidence between each node and its labeled multi-hop neighborhoods.

Results

	Method	Cora	Citeseer	Pubmed
GCNs	GCN [19]	81.5	70.3	79.0
	GAT [32]	83.0±0.7	72.5±0.7	79.0±0.3
	APPNP [20]	83.8±0.3	71.6± 0.5	79.7 ± 0.3
	Graph U-Net [11]	84.4±0.6	73.2±0.5	79.6±0.2
	SGC [36]	81.0 ±0.0	71.9 ± 0.1	78.9 ± 0.0
	MixHop [1]	81.9± 0.4	71.4±0.8	80.8±0.6
	GMNN [28]	83.7	72.9	81.8
	GraphNAS [12]	84.2±1.0	73.1±0.9	79.6±0.4
Sampling GCNs	GraphSAGE [16]	78.9±0.8	67.4±0.7	77.8±0.6
	FastGCN [7]	81.4±0.5	68.8±0.9	77.6±0.5
Reg. GCNs	VBAT [10]	83.6±0.5	74.0±0.6	79.9±0.4
	G^3NN [24]	82.5±0.2	74.4±0.3	77.9 ±0.4
	GraphMix [33]	83.9±0.6	74.5±0.6	81.0±0.6
	DropEdge [29]	82.8	72.3	79.6
	GRAND	**85.4±0.4**	**75.4±0.4**	**82.7±0.6**



Instead of the marginal improvements by conventional GNN baselines over GCN, ***GRAND*** *achieves* ***much more significant*** *performance lift in* ***all three datasets***!

- W. Feng, J. Zhang, Y. Dong, Y. Han, H. Luan, Q. Xu, Q. Yang, E. Kharlamov, and J. Tang. Graph Random Neural Networks for Semi-Supervised Learning on Graphs. NeurIPS'20. https://arxiv.org/abs/2005.11079
- Code & data for Grand: https://github.com/Grand20/grand

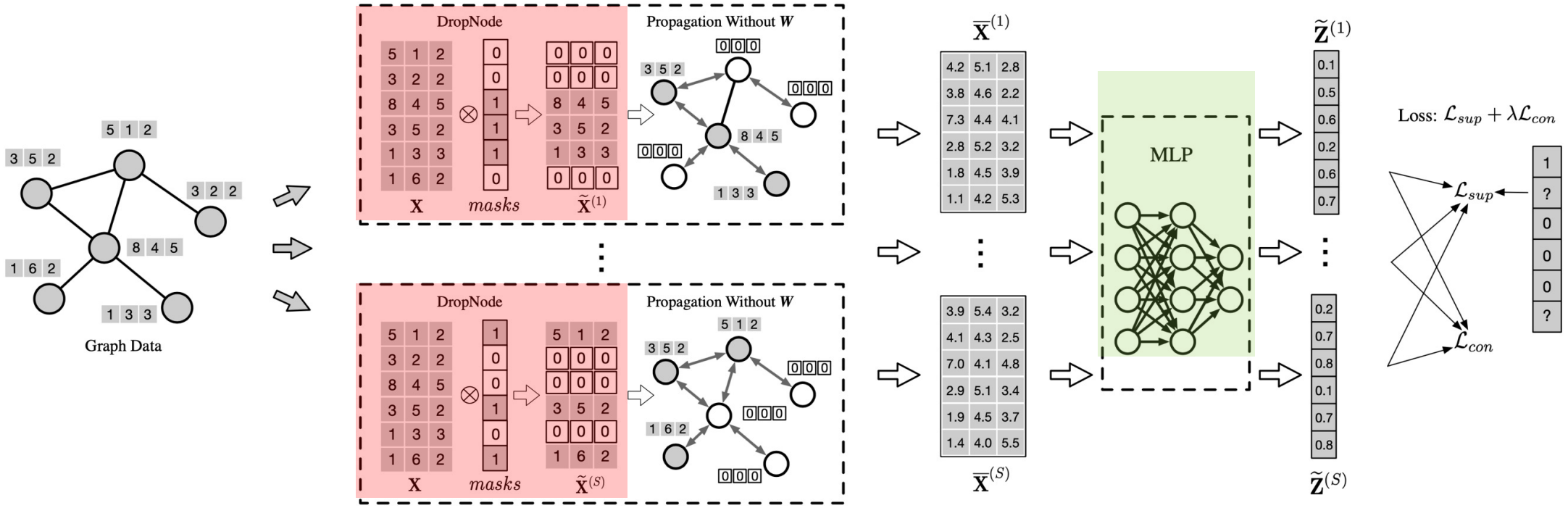
Results

Table 5: Results on large datasets.

Method	Cora Full	Coauthor CS	Coauthor Physics	Amazon Computer	Amazon Photo	Citation CS
GCN	62.2 ± 0.6	91.1 ± 0.5	92.8 ± 1.0	82.6 ± 2.4	91.2 ± 1.2	49.9 ± 2.0
GAT	51.9 ± 1.5	90.5 ± 0.6	92.5 ± 0.9	78.0 ± 19.0	85.7 ± 20.3	49.6 ± 1.7
GRAND	**63.5 ±0.6**	**92.9 ± 0.5**	**94.6 ± 0.5**	**85.7 ± 1.8**	**92.5 ± 1.7**	**52.8 ± 1.2**

More experiments on larger graph datasets

Results



GRAND_dropout	84.9±0.4	75.0±0.3	81.7±1.0
GRAND_GCN	84.5±0.3	74.2±0.3	80.0±0.3
GRAND_GAT	84.3±0.4	73.2± 0.4	79.2±0.6
GRAND	**85.4±0.4**	**75.4±0.4**	**82.7±0.6**

Evaluation of the design choices in GRAND

Results

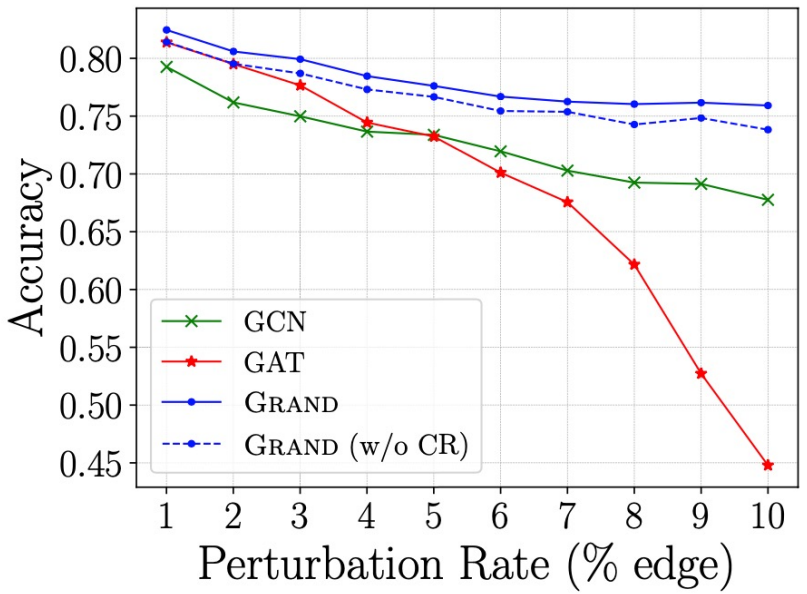
Method	Cora	Citeseer	Pubmed
GCN [19]	81.5	70.3	79.0
GAT [32]	83.0±0.7	72.5±0.7	79.0±0.3
APPNP [20]	83.8±0.3	71.6± 0.5	79.7 ± 0.3
Graph U-Net [11]	84.4±0.6	73.2±0.5	79.6±0.2
SGC [36]	81.0 ±0.0	71.9 ± 0.1	78.9 ± 0.0
MixHop [1]	81.9± 0.4	71.4±0.8	80.8±0.6
GMNN [28]	83.7	72.9	81.8
GraphNAS [12]	84.2±1.0	73.1±0.9	79.6±0.4
DropEdge [29]	82.8	72.3	79.6
w/o CR	84.4±0.5	73.1±0.6	80.9±0.8
w/o mDN	84.7±0.4	74.8±0.4	81.0±1.1
w/o sharpening	84.6±0.4	72.2±0.6	81.6±0.8
w/o CR & DN	83.2±0.5	70.3±0.6	78.5±1.4

Ablation Study

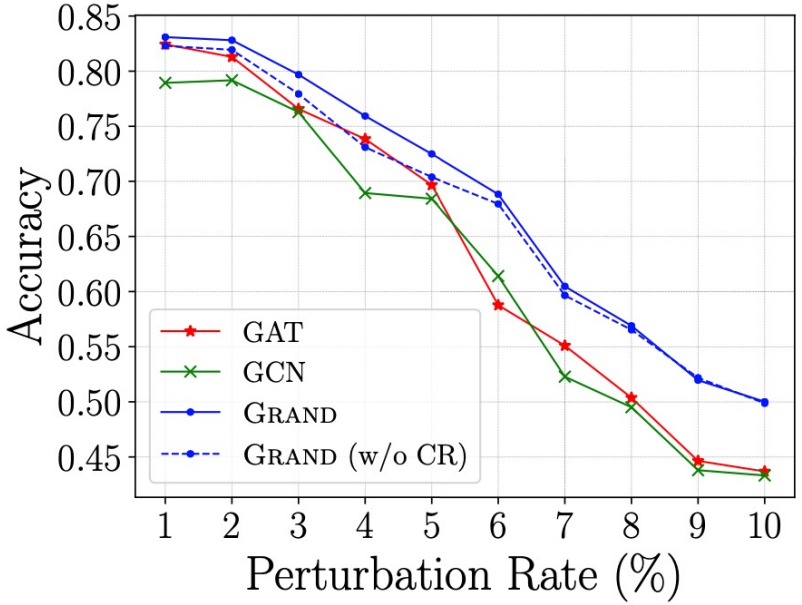
1. Each of the designed components contributes to the success of GRAND.
2. GRAND w/o consistency regularization outperforms almost *all 8 non-regularization based GCNs* & *DropEdge*

Random Propagation
vs.
Feature Propagation & Non-Linear Transformation

Results



(a) Random Attack



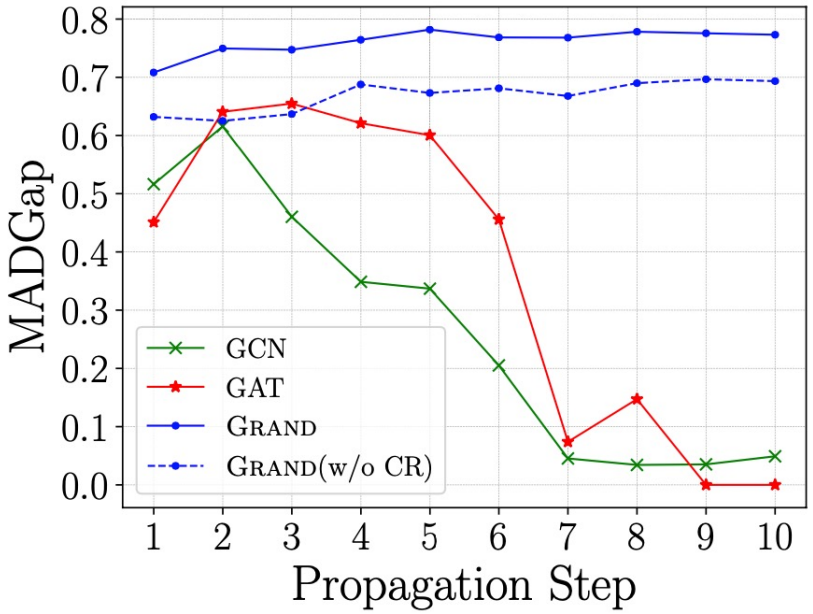
(b) Metattack

Robustness

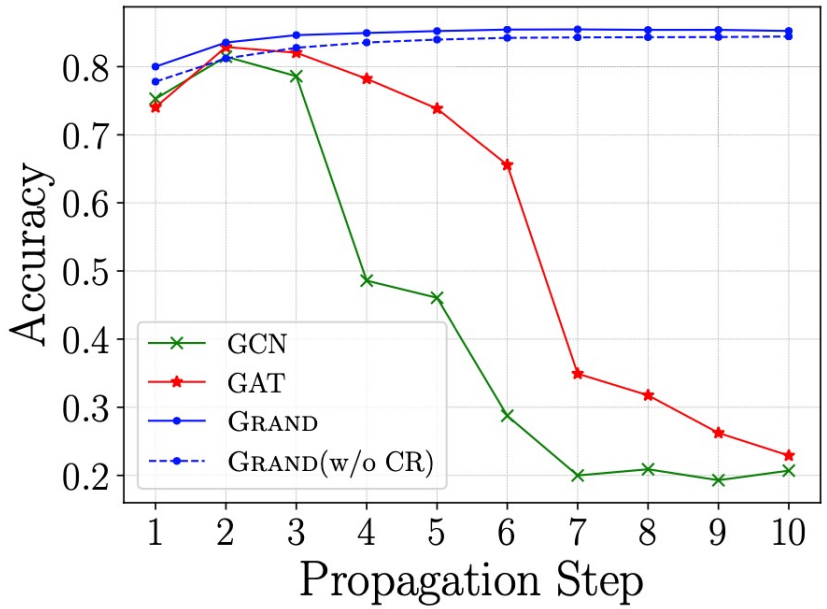
1. GRAND (with or w/o) consistency regularization is more robust than GCN and GAT.

Random Propagation vs. Feature Propagation & Non-Linear Transformation

Results



(a) MADGap



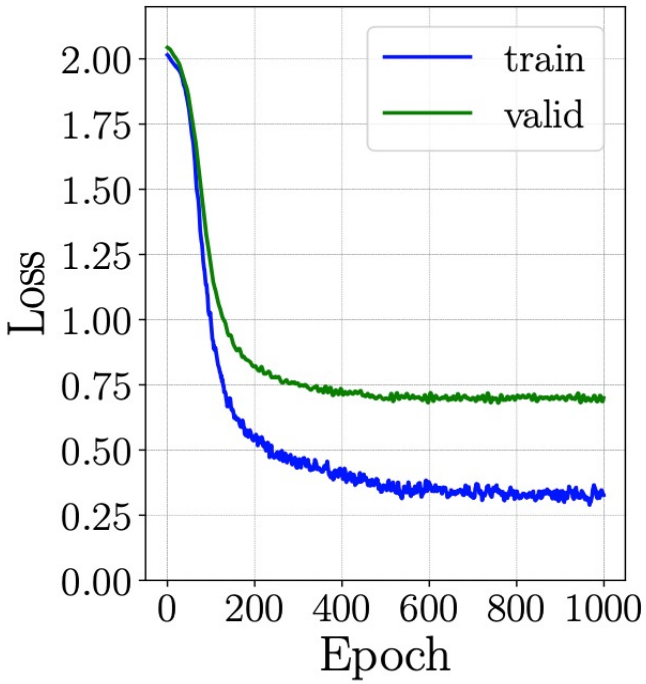
(b) Classification Results

Over-Smoothing

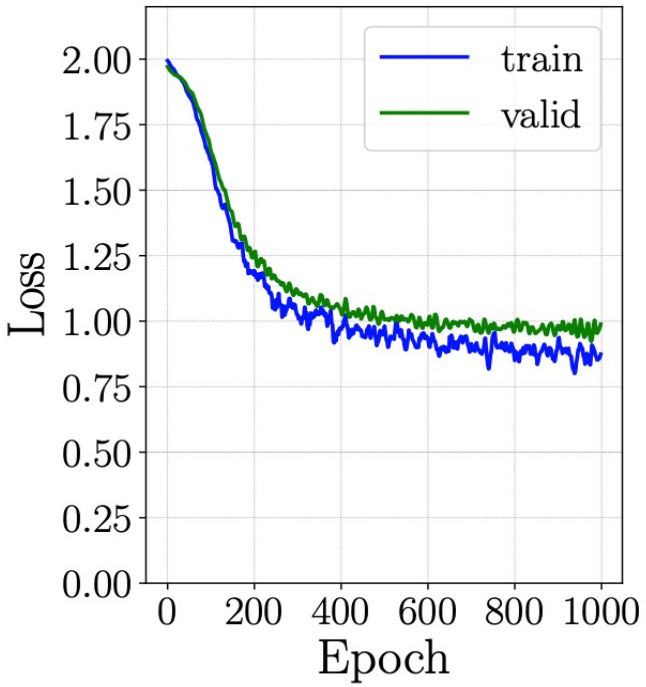
1. GRAND is powerful to relieve over-smoothing, while GCN & GAT are vulnerable to it

Random Propagation vs. Feature Propagation & Non-Linear Transformation

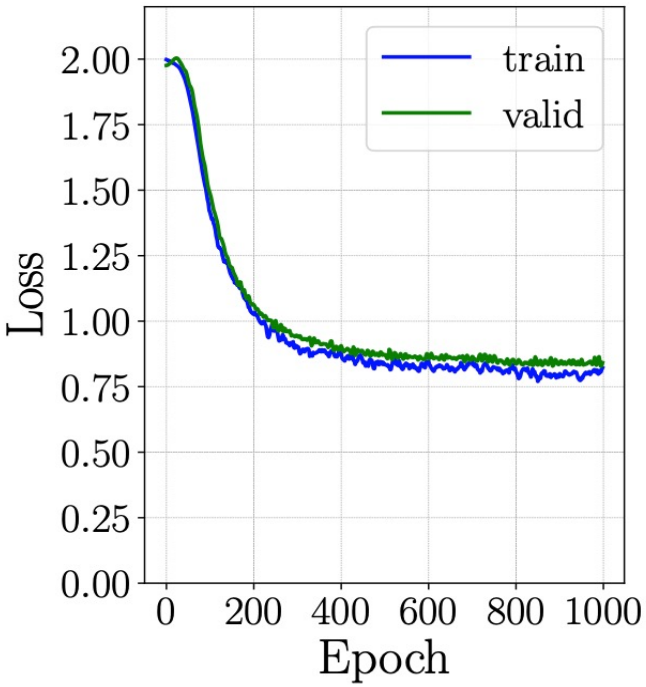
Results



(a) Without CR and RP



(b) Without CR



(c) GRAND

Generalization

1. Both the random propagation and consistency regularization improve GRAND's generalization capability

Graph Robustness Benchmark: Benchmarking the Adversarial Robustness of Graph Machine Learning

Qinkai Zheng[†], Xu Zou[†], Yuxiao Dong[‡*], Yukuo Cen[†], Da Yin[†], Jiarong Xu[°], Yang Yang[◇], Jie Tang[†‡]

[†] Department of Computer Science and Technology, Tsinghua University

[‡] Microsoft Research, Redmond [°] Fudan University [◇] Zhejiang University

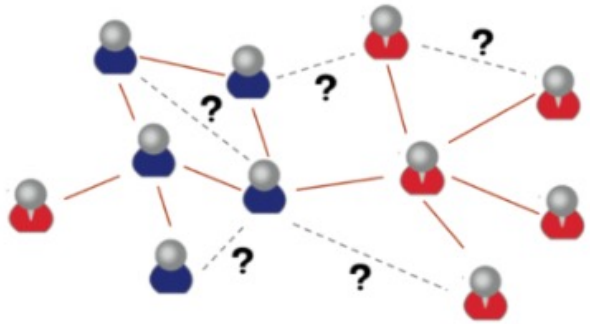
GRB

GRAPH ROBUSTNESS BENCHMARK

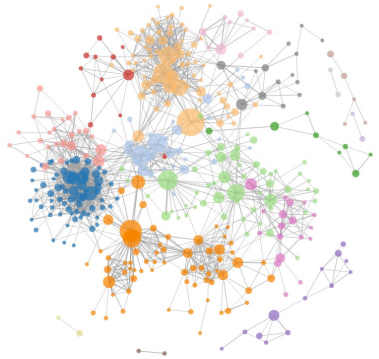
Homepage: https://cogdl.ai/grb/home **Code**: https://github.com/THUDM/grb

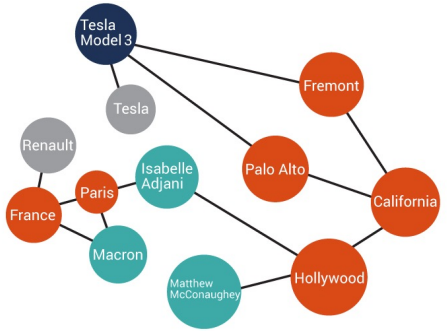
Adversarial Robustness in Graph Machine Learning

- Graph machine learning (GML) has made great progress.



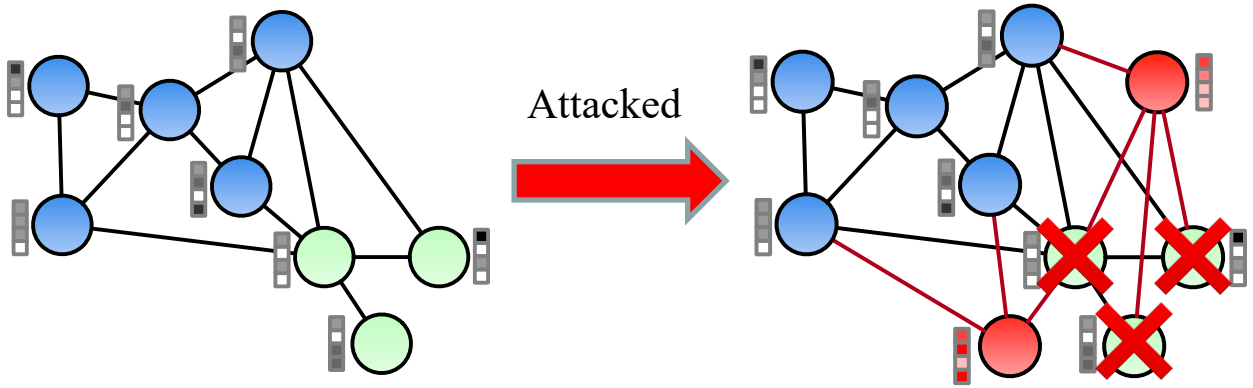
Social network


Academic network



Knowledge graph

- However, GML models face big threats from adversarial attacks.

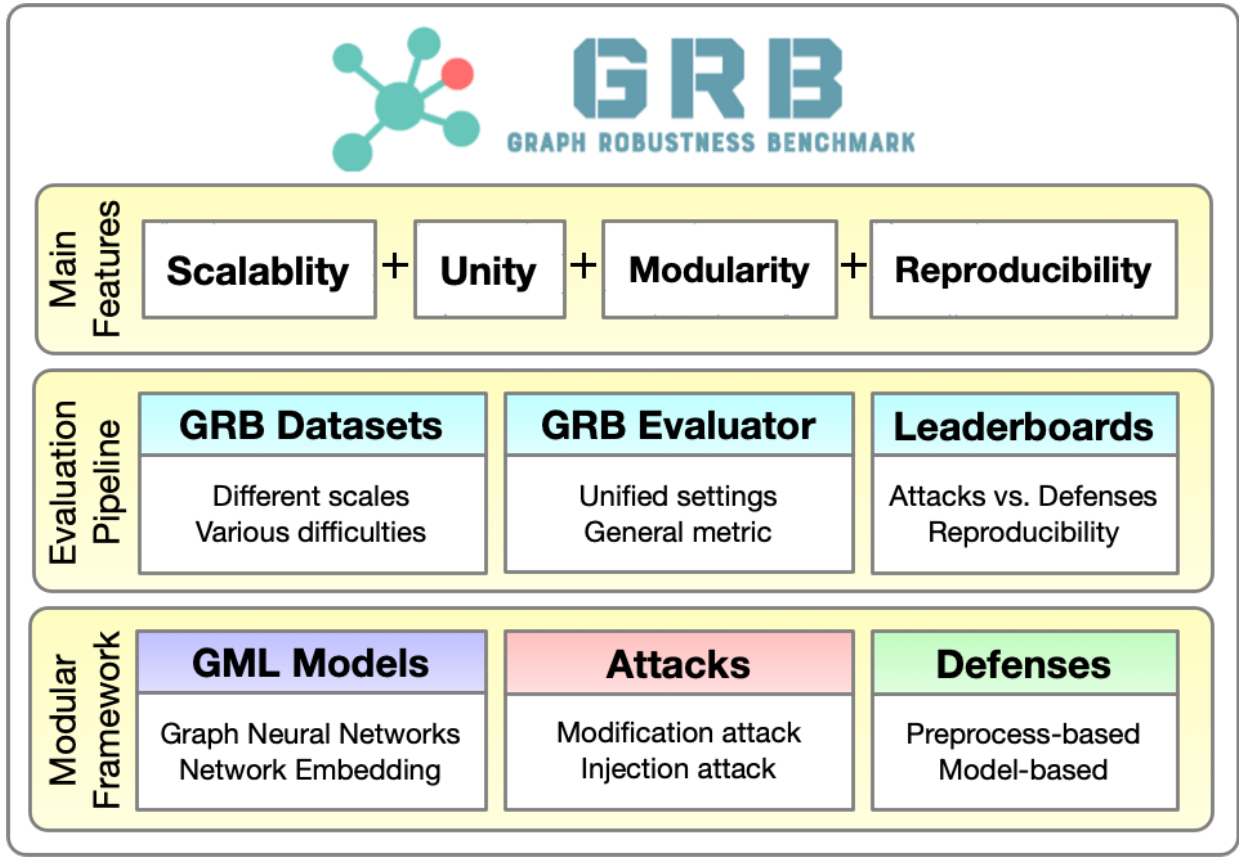


Adversarial Robustness:
The ability to maintain performance under potential adversarial attacks.

1. X. Zou, Q. Zheng, Y. Dong, X. Guan, E. Kharlamov, J. Lu, and J. Tang. TDGIA: Effective Injection Attacks on Graph Neural Networks. **KDD'21.**
2. Zheng et al., Graph Robustness Benchmark: Benchmarking the Adversarial Robustness of Graph Machine Learning, **NeurIPS'21**.

Code & data for Grand: https://cogdl.ai/grb/home https://github.com/THUDM/grb

Graph Robustness Benchmark (GRB)



- **Elaborated datasets**
  - From small- to large- scale graphs.
  - Robust-specific splitting and normalization.
- **Unified evaluation protocol**
  - Unified settings and general metrics.
  - Unified evaluation pipeline.
- **Refined attack/defense scenarios**
  - More practical scenarios.
  - More realistic assumptions.
- **Modular coding framework**
  - Easy implementation.
  - Esurance of reproducibility.

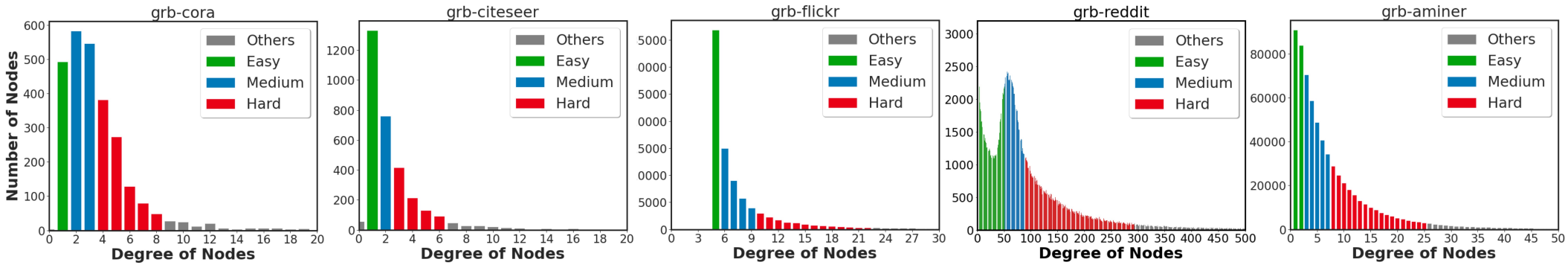
Altogether, **GRB** serves as a *scalable*, *unified*, *modular*, and *reproducible* benchmark.

GRB Datasets

- **Scalability**: From small to large graphs with hundreds of thousands of nodes (millions of edges).

Dataset	Scale	#Nodes	#Edges	#Feat.	#Classes	Feat. Range	Feat. Range (normalized)
grb-cora	Small	2,680	5,148	302	7	[-2.30, 2.40]	[-0.94, 0.94]
grb-citeseer	Small	3,191	4,172	768	6	[-4.55, 1.67]	[-0.96, 0.89]
grb-flickr	Medium	89,250	449,878	500	7	[-0.90, 269.96]	[-0.47, 1.00]
grb-reddit	Large	232,965	11,606,919	602	41	[-28.19, 120.96]	[-0.98, 0.99]
grb-aminer	Large	659,574	2,878,577	100	18	[-1.74, 1.62]	[-0.93, 0.93]

- **Robust-specific splitting**: Various difficulties according to the *average degree* of target nodes.



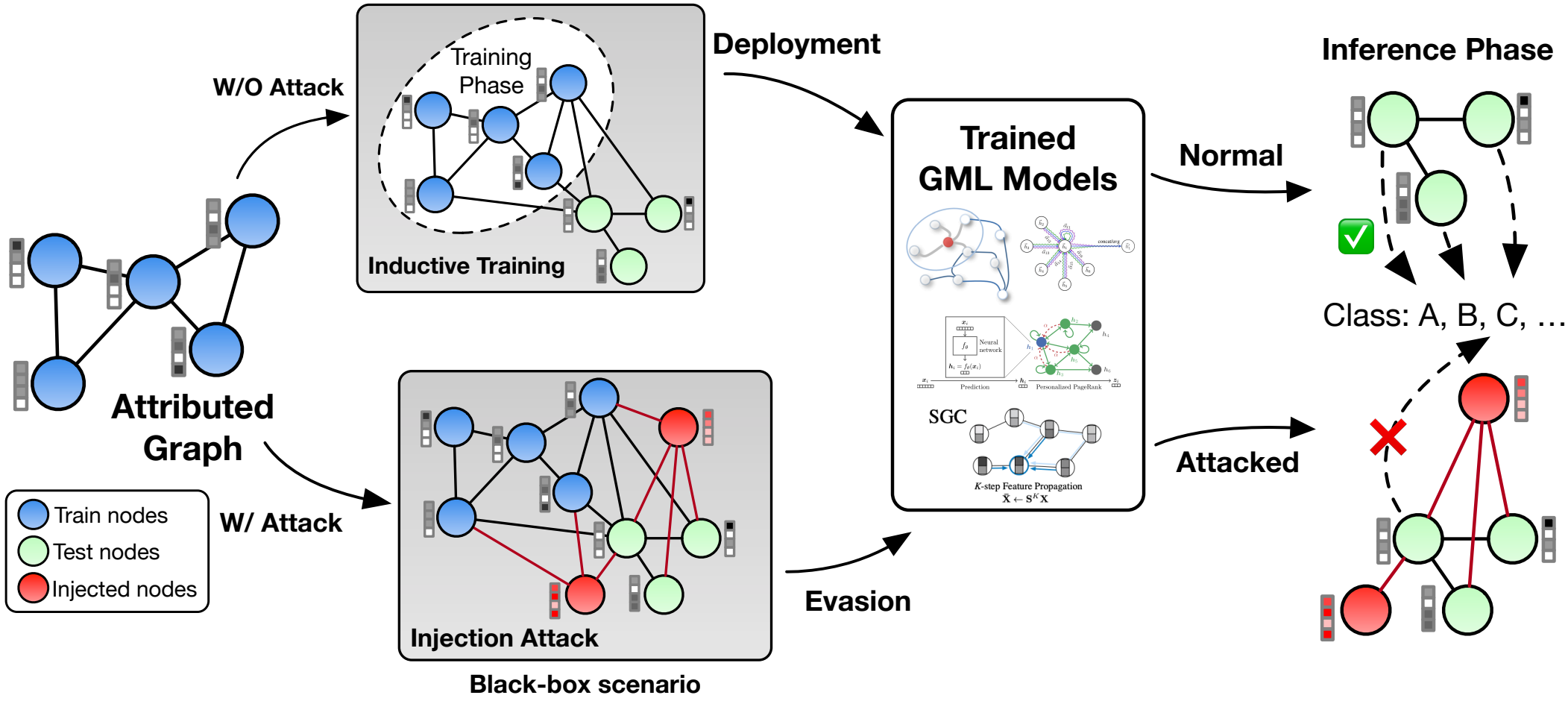
- **Feature normalization:** Unify values to the same scale, $\mathcal{F} = \frac{2}{\pi} \arctan\left(\frac{\mathcal{F} - \text{mean}(\mathcal{F})}{\text{std}(\mathcal{F})}\right)$

1. X. Zou, Q. Zheng, Y. Dong, X. Guan, E. Kharlamov, J. Lu, and J. Tang. TDGIA: Effective Injection Attacks on Graph Neural Networks. **KDD'21.**
2. Zheng et al., Graph Robustness Benchmark: Benchmarking the Adversarial Robustness of Graph Machine Learning, **NeurIPS'21**.

Code & data for Grand: https://cogdl.ai/grb/home https://github.com/THUDM/grb

GRB Attack vs. Defense Scenario

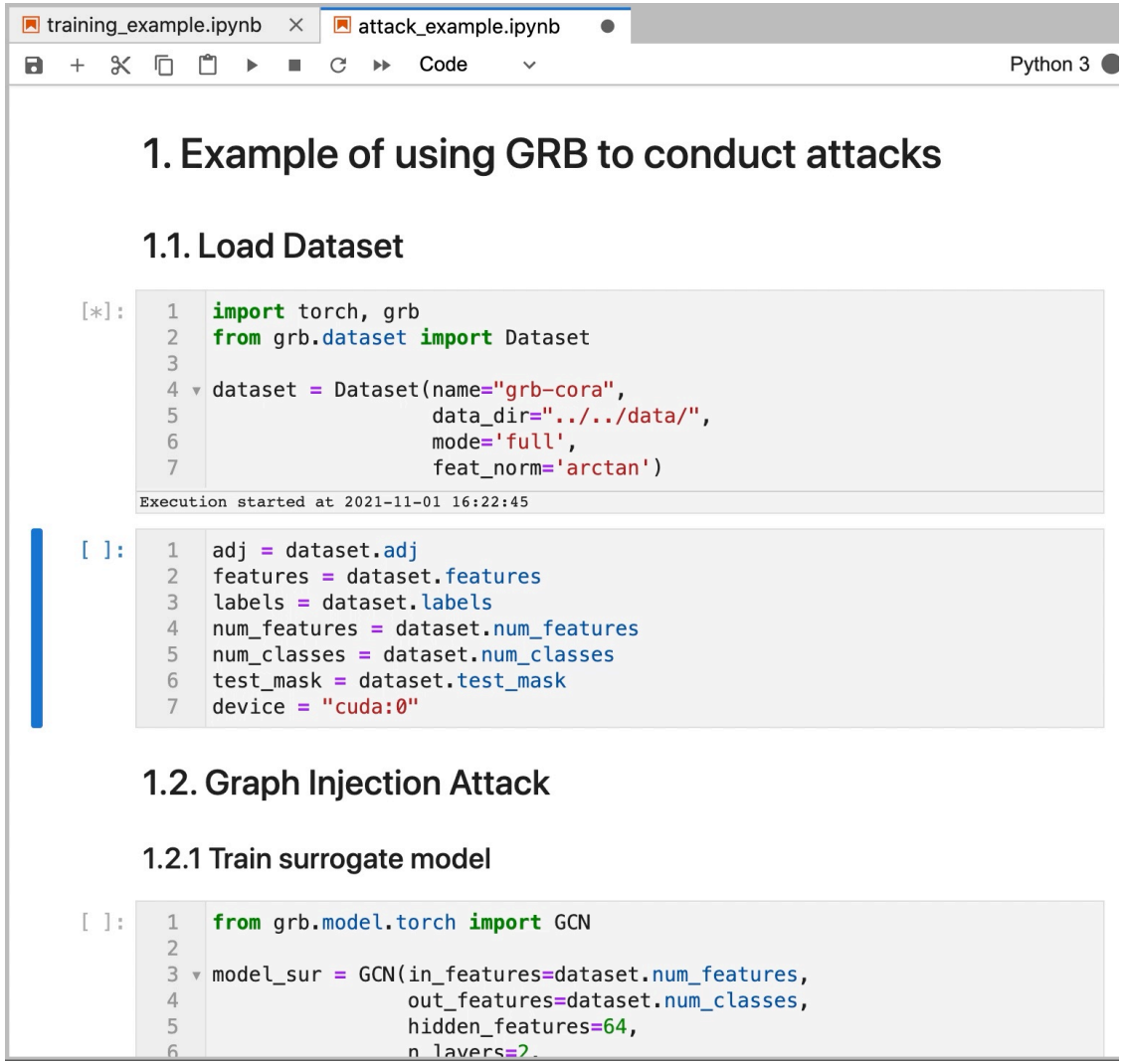
- Example of *graph injection* scenario (*black-box, inductive, evasion*)



Constraints: limited number of injected nodes each with limited edges + constrained feature range.

GRB Modular Coding Framework

GRB Modular Coding Framework				
Dataset	Dataloader	Dataset		
	Preprocessing	Splitting		
		Feature normalization		
Model	GML Models	GCN, SAGE, GAT, GIN, APPNP, ...		
Attack	Modification	RND, DICE, FLIP, STACK, ...		
	Injection	RND, FGSM, SPEIT, TDGIA, ...		
Defense	Preprocess-based	GNN-SVD, GNNGuard, ...		
	Model-based	RobustGNN, Adversarial training, ...		
Evalutor	General Metrics	Avg. 3-Min/Max Acc., Weighed Acc., ...		
Others	Pipeline	AutoML	Visualizer	...
Backend	Pytorch	CogDL	DGL	



Example of using **GRB**

Example of GRB Leaderboard

grb-aminer leaderboard (Top 5 ATK. vs. Top 10 DEF.) in *graph injection* scenario

	Attack \ Defenses		1 GAT+AT	2 R-GCN+AT	3 SGCN+LN	4 R-GCN	5 GCN+LN	6 GATLN	7 GIN+LN	8 TAGCN+LN	9 TAGCN+AT	10 GAT	Avg. Acc.	Avg. 3-Max Acc.	Weighted Acc.
1	TDGIA	F	67.69±0.03	63.62±0.32	62.20±0.15	61.99±0.22	60.38±1.46	59.69±1.57	59.59±0.42	59.06±1.75	57.24±5.04	56.63±6.75	60.81±1.71	64.52±2.32	65.74±0.21
2	SPEIT	F	68.04±0.03	64.05±0.04	64.84±0.04	64.06±0.04	65.51±0.02	64.02±0.04	63.11±0.02	62.59±0.04	63.77±0.06	63.58±0.06	64.36±0.02	66.13±1.38	66.89±0.02
3	RND	F	67.72±0.04	64.98±0.02	65.31±0.04	64.45±0.04	66.17±0.02	67.54±0.04	64.36±0.06	64.33±0.03	66.42±0.03	66.23±0.04	65.75±0.02	67.23±0.58	67.34±0.03
4	PGD	F	68.01±0.02	65.41±0.01	65.54±0.03	65.05±0.03	66.22±0.02	66.49±0.04	64.63±0.04	64.82±0.04	66.32±0.02	66.14±0.04	65.86±0.01	66.94±0.76	67.37±0.02
5	FGSM	F	68.00±0.02	65.41±0.02	65.54±0.04	65.05±0.04	66.22±0.02	66.50±0.06	64.65±0.04	64.82±0.03	66.34±0.03	66.15±0.06	65.87±0.01	66.95±0.75	67.37±0.01
	W/O Attack	F	67.93±0.00	65.76±0.00	66.68±0.00	65.85±0.00	66.20±0.00	68.47±0.00	65.59±0.00	64.91±0.00	67.08±0.00	68.02±0.00	66.65±0.00	68.14±0.24	68.11±0.00
	Avg. Acc.	F	67.90±0.01	64.87±0.05	65.02±0.03	64.41±0.04	65.12±0.25	65.45±0.26	63.65±0.07	63.42±0.29	64.53±0.84	64.46±1.13	-	-	-
	Avg. 3-Min	F	67.78±0.02	64.22±0.11	64.12±0.06	63.50±0.08	64.02±0.49	63.39±0.53	62.35±0.14	61.99±0.58	62.44±1.69	62.11±2.26	-	-	-
	Weighted Acc.	F	67.73±0.03	63.96±0.21	63.19±0.10	62.80±0.15	62.18±0.98	61.58±1.05	61.00±0.28	60.54±1.18	59.82±3.38	59.37±4.53	-	-	-

- **Metric for Attacks**: Avg. Acc., Avg. 3-Max Acc., Weighted Acc.
- **Metric for Defenses**: Avg. Acc., Avg. 3-Min Acc., Weighted Acc.
- **Avg. 3-Min/Max Acc.**: average accuracy of three most effective attacks or three most robust models.
- **Weighted Acc.:** attach higher weight for more effective attacks or more robust models.
- **Ranking:** Ranked by Weighted Acc., red and blue indicate the best results of attacks/defenses respectively.
- More results for other difficulties, datasets and scenarios can be found on our website.

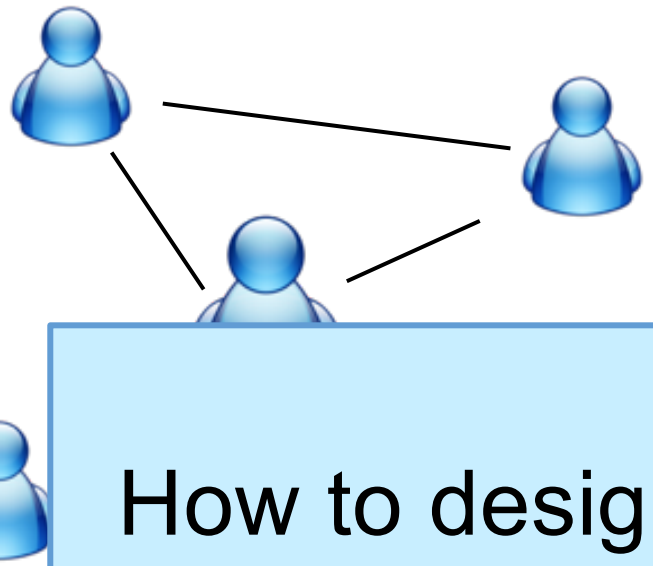
● **GRB** focuses on *generally more effective* methods, rather than comparing a single pair of attack/defense.

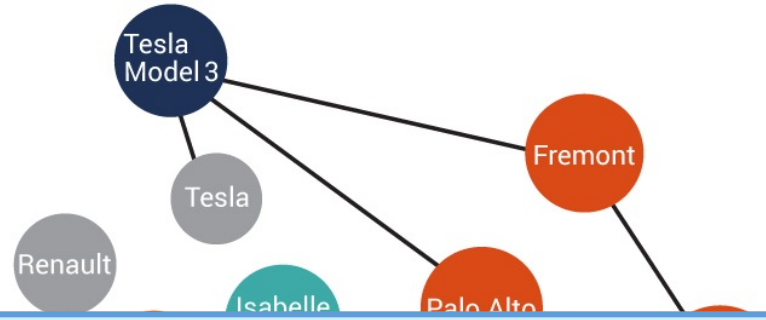
1. X. Zou, Q. Zheng, Y. Dong, X. Guan, E. Kharlamov, J. Lu, and J. Tang. TDGIA: Effective Injection Attacks on Graph Neural Networks. **KDD'21.**
2. Zheng et al., Graph Robustness Benchmark: Benchmarking the Adversarial Robustness of Graph Machine Learning, **NeurIPS'21**.

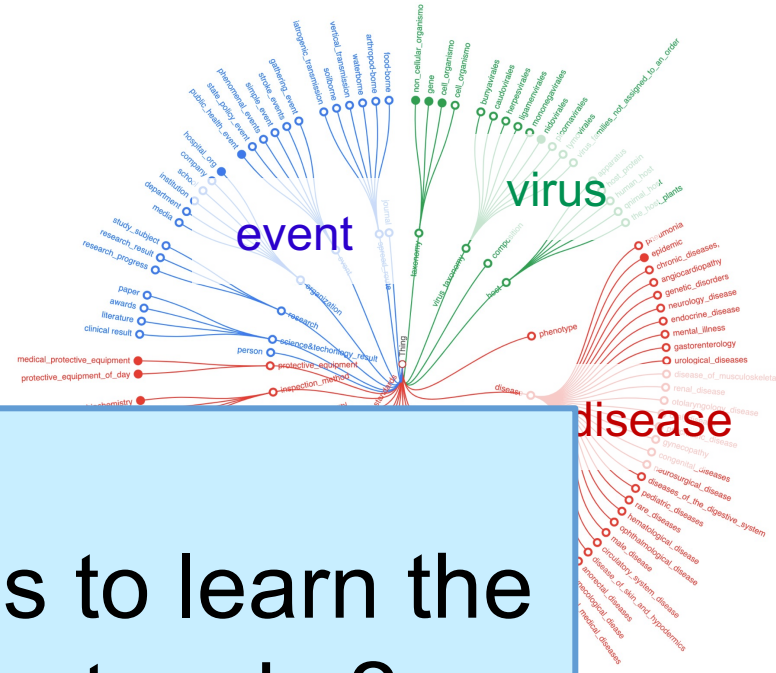
Code & data for Grand: https://cogdl.ai/grb/home https://github.com/THUDM/grb

GCC: Graph Contrastive Coding for Graph Neural Network Pre-Training

Networked Data







Question:

How to design machine learning models to learn the universal structural patterns across networks?

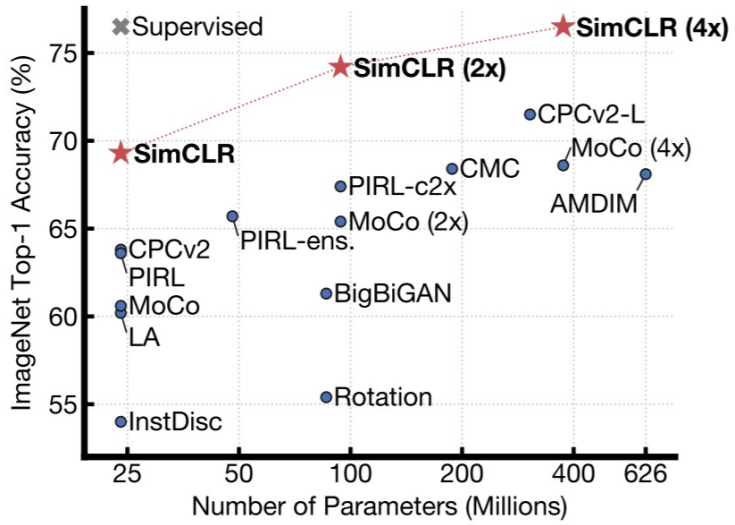
Social Network

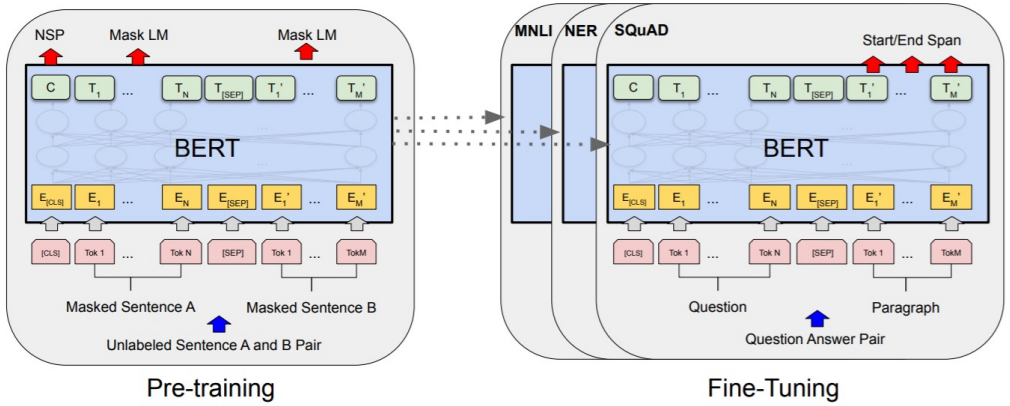
Knowledge Graph

COVID Graph

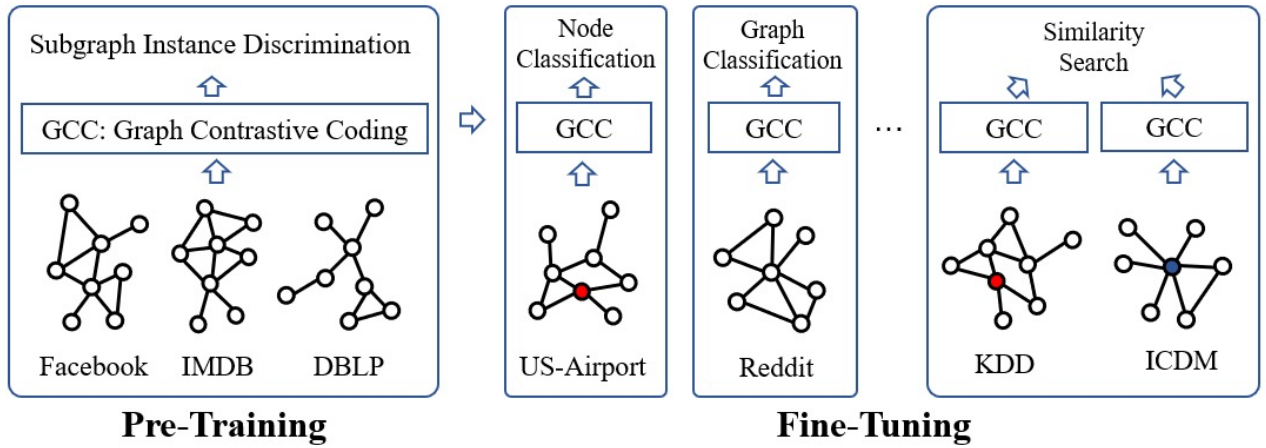
Pre-training and Fine-tuning

CV: MoCo, SimCLR





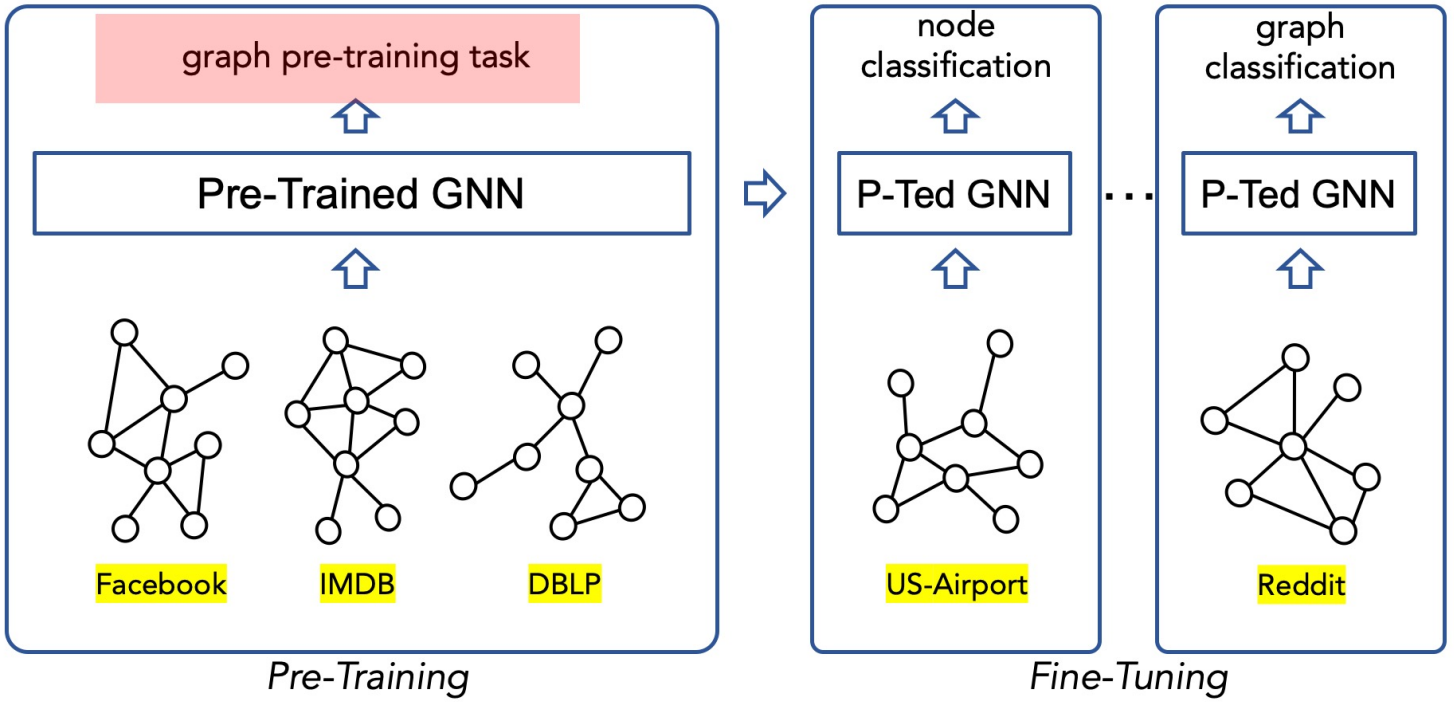
NLP: BERT



Graph Learning
GCC

GNN Pre-Training

- Graph pre-training setting:
  - To pre-train from some graphs
  - To fine-tune for unseen tasks on unseen graphs



- How to do this?
  - Model level: GNNs?
  - Pre-training task: **self-supervised** tasks across graphs?

GNN Pre-Training across Networks

- What are the requirements?
  - **structural similarity**, it maps vertices with similar local network topologies close to each other in the vector space
  - **transferability**, it is compatible with vertices and graphs unseen by the pre-training algorithm

1. Jiezhong Qiu et al. GCC: Graph Contrastive Coding for Graph Neural Network Pre-Training. **KDD** 2020.
2. Code & Data for GCC: https://github.com/THUDM/GCC

GNN Pre-Training across Networks

- The Idea: Contrastive learning
  - **pre-training task**: instance discrimination
  - **InfoNCE objective**: output instance representations that are capable of capturing the similarities between instances

$$\mathcal{L} = -\log \frac{\exp\left(\boldsymbol{q}^\top \boldsymbol{k}_+ / \tau\right)}{\sum_{i=0}^{K} \exp\left(\boldsymbol{q}^\top \boldsymbol{k}_i / \tau\right)}$$

- query instance x^q
- query $\boldsymbol{q}$ (embedding of x^q), i.e., $\boldsymbol{q} = f(x^q)$
- dictionary of keys $\{\boldsymbol{k}_0, \boldsymbol{k}_1, \cdots, \boldsymbol{k}_K\}$
- key $\boldsymbol{k} = f(x^k)$

- Contrastive learning for graphs?
  - Q1: How to define instances in graphs?
  - Q2: How to define (dis) similar instance pairs in and across graphs?
  - Q3: What are the proper graph encoders?

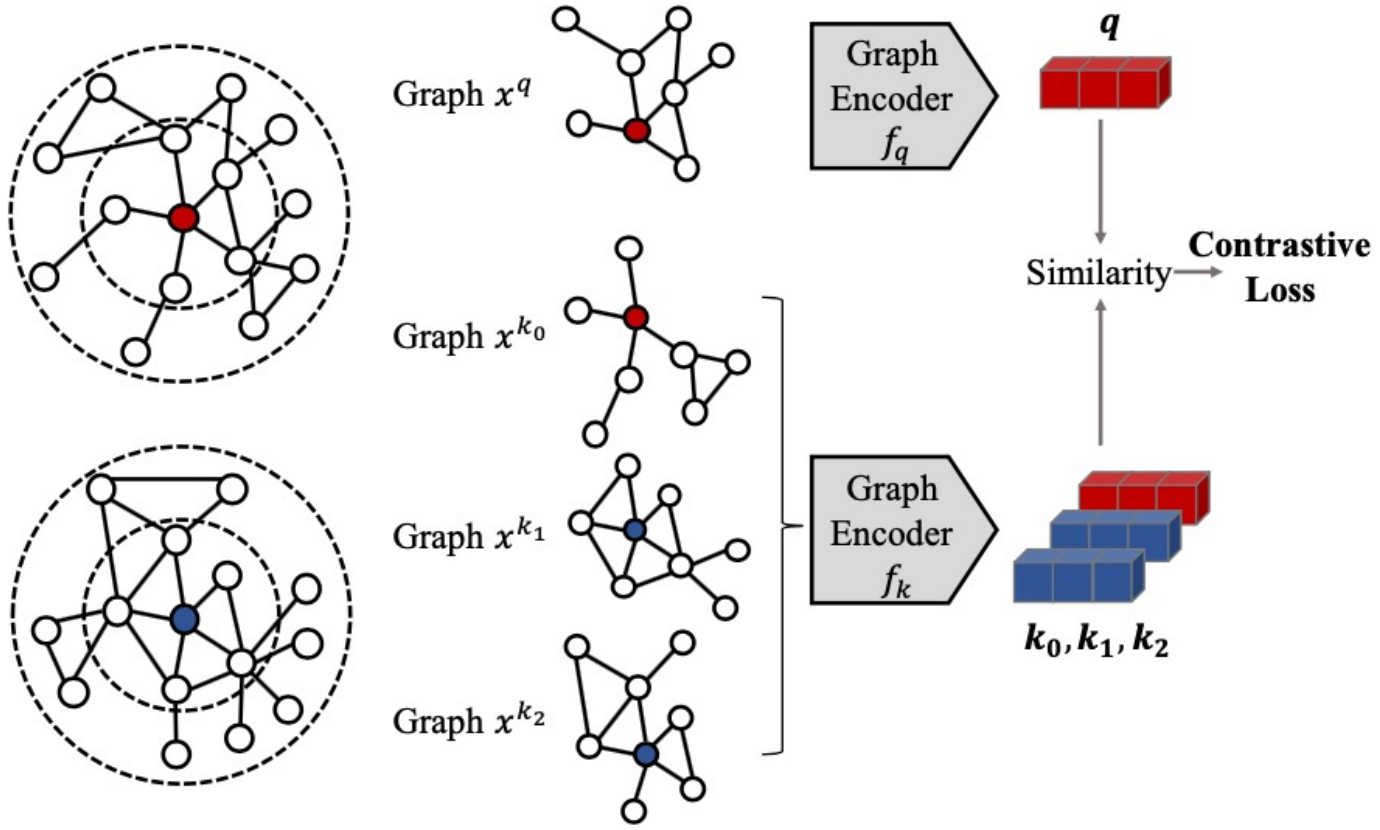
1. Z. Wu, Y. Xiong, S. X Yu, and D. Lin. Unsupervised feature learning via non-parametric instance discrimination. In CVPR '18.
2. K. He, H. Fan, Y. Wu, S. Xie, and R. Girshick. Momentum contrast for unsupervised visual representation learning. In CVPR '20.

Graph Contrastive Coding (GCC)

- Contrastive learning for graphs
  - Q1: How to define instances in graphs?
  - Q2: How to define (dis) similar instance pairs in and across graphs?
  - Q3: What are the proper graph

$$\mathcal{L} = -\log \frac{\exp\left(\boldsymbol{q}^\top \boldsymbol{k}_+ / \tau\right)}{\sum_{i=0}^{K} \exp\left(\boldsymbol{q}^\top \boldsymbol{k}_i / \tau\right)}$$

Subgraph instance discrimination



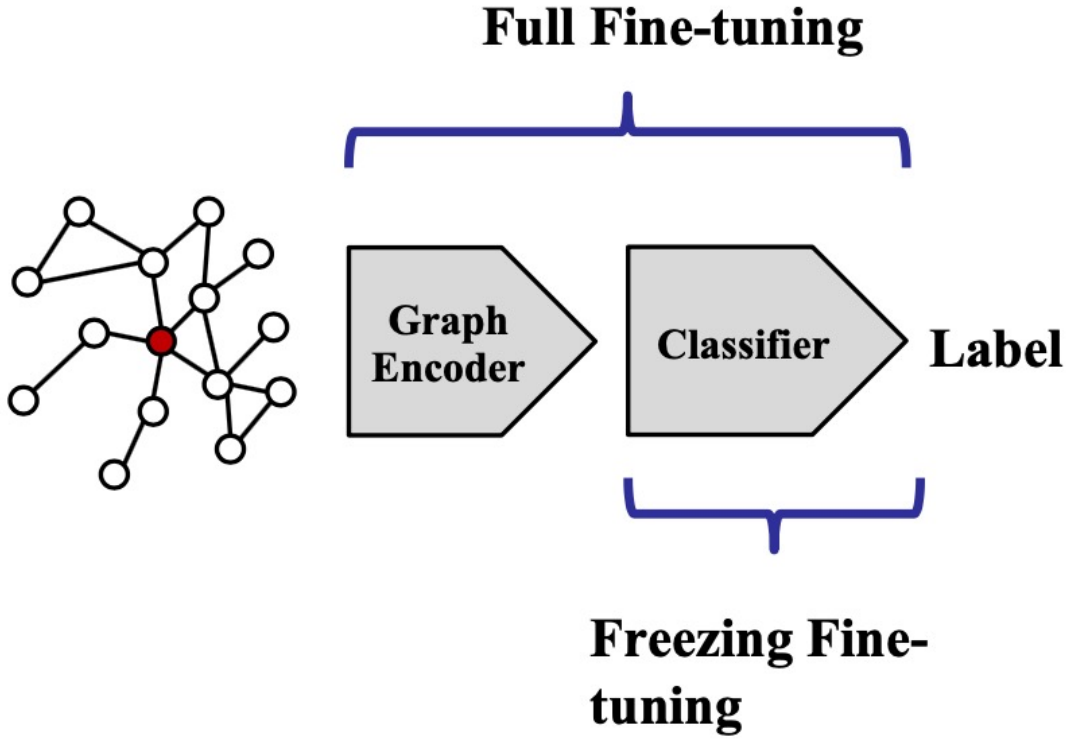
GCC Pre-Training / Fine-Tuning

- pre-train on six graphs

Code & Data for GCC:
https://github.com/THUDM/GCC

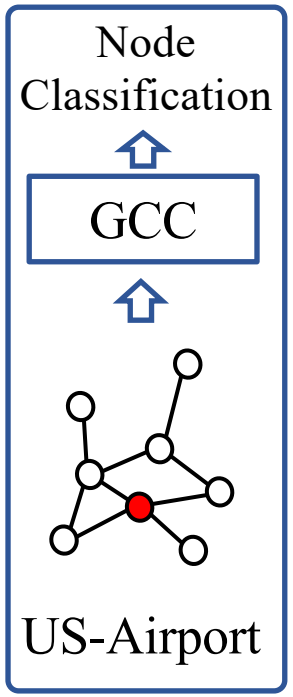
Dataset	Academia	DBLP (SNAP)	DBLP (NetRep)	IMDB	Facebook	LiveJournal
$\|V\|$	137,969	317,080	540,486	896,305	3,097,165	4,843,953
$\|E\|$	739,384	2,099,732	30,491,458	7,564,894	47,334,788	85,691,368

- fine-tune on **different** graphs
  - US-Airport & AMiner academic graph
    - Node classification
  - COLLAB, RDT-B, RDT-M, & IMDB-B/M
    - Graph classification
  - AMiner academic graph
    - Similarity search
- The base GNN
  - Graph Isomorphism Network (GIN)



Result 1: Node Classification

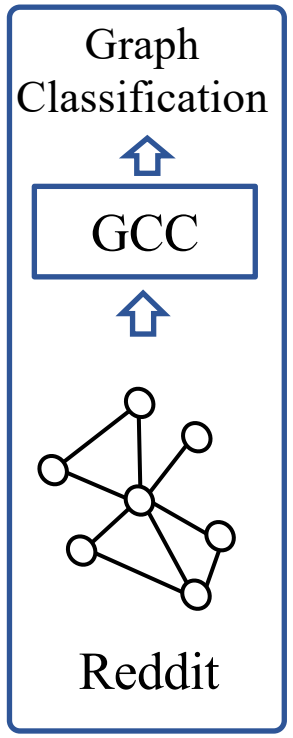
- Setup
  - US-Airport
  - AMiner academic graph



Datasets	US-Airport	H-index
$\|V\|$	1,190	5,000
$\|E\|$	13,599	44,020
ProNE	62.3	69.1
GraphWave	60.2	70.3
Struc2vec	**66.2**	> 1 Day
GCC (E2E, freeze)	64.8	**78.3**
GCC (MoCo, freeze)	65.6	75.2
GCC (rand, full)	64.2	76.9
GCC (E2E, full)	**68.3**	80.5
GCC (MoCo, full)	67.2	**80.6**

Result 2: Graph Classification

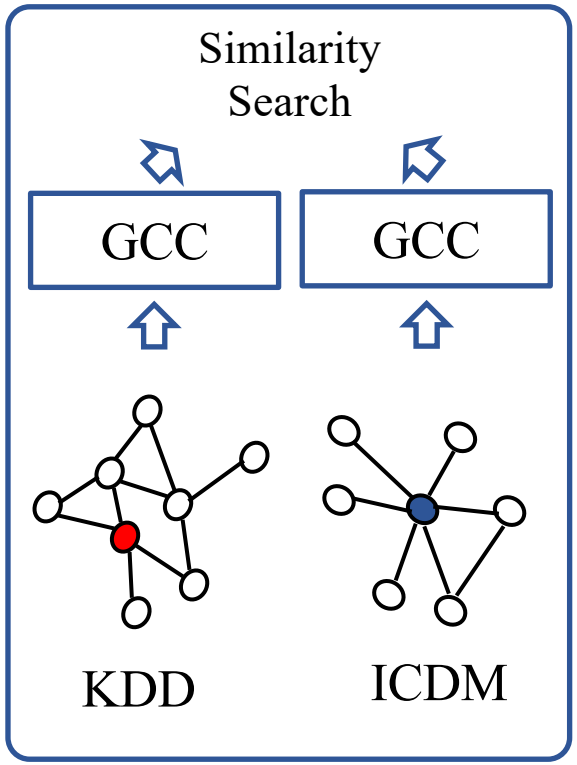
- Setup
  - COLLAB, RDT-B, RDT-M, & IMDB-B, IMDB-M



Datasets	IMDB-B	IMDB-M	COLLAB	RDT-B	RDT-M
# graphs	1,000	1,500	5,000	2,000	5,000
# classes	2	3	3	2	5
Avg. # nodes	19.8	13.0	74.5	429.6	508.5
DGK	67.0	44.6	73.1	78.0	41.3
graph2vec	71.1	50.4	–	75.8	47.9
InfoGraph	**73.0**	**49.7**	–	82.5	53.5
GCC (E2E, freeze)	71.7	49.3	74.7	87.5	52.6
GCC (MoCo, freeze)	72.0	49.4	**78.9**	**89.8**	**53.7**
DGCNN	70.0	47.8	73.7	–	–
GIN	**75.6**	**51.5**	80.2	**89.4**	**54.5**
GCC (rand, full)	**75.6**	50.9	79.4	87.8	52.1
GCC (E2E, full)	70.8	48.5	79.0	86.4	47.4
GCC (MoCo, full)	73.8	50.3	**81.1**	87.6	53.0

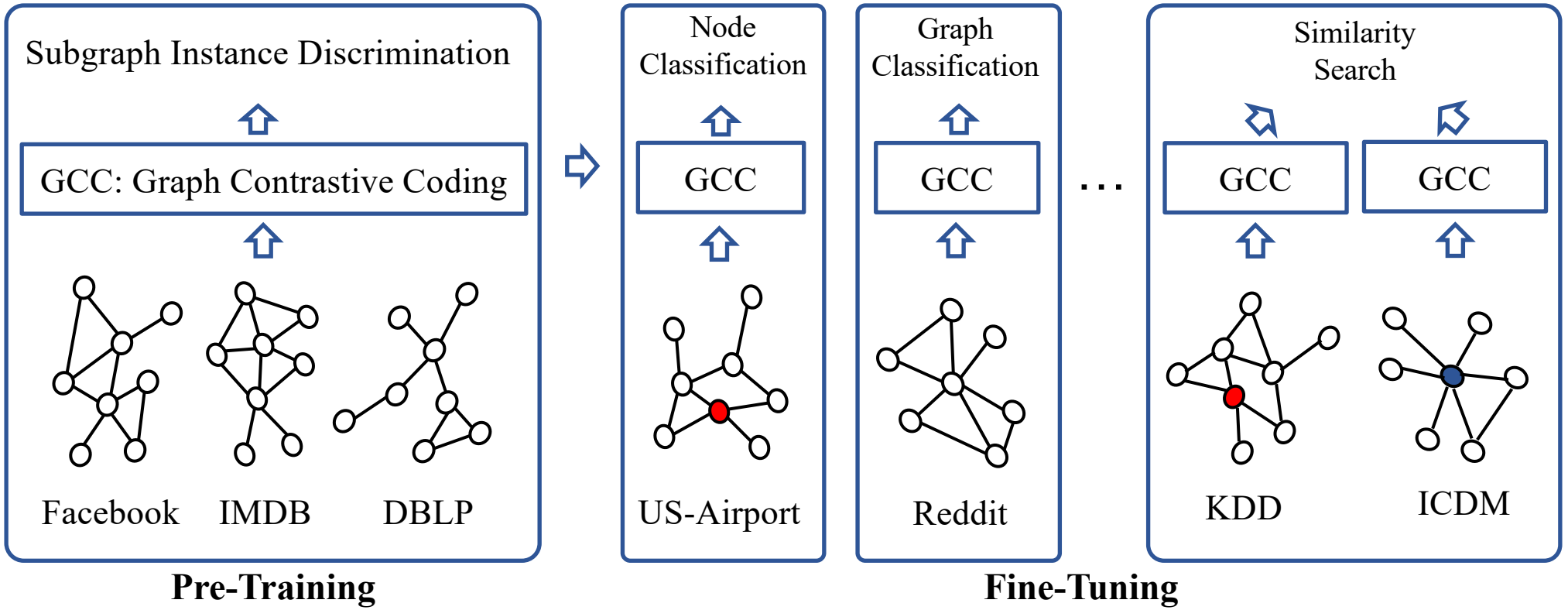
Result 3: Top-k Similarity Search

- Setup
  - AMiner academic graph



	KDD-ICDM		SIGIR-CIKM		SIGMOD-ICDE	
$\|V\|$	2,867	2,607	2,851	3,548	2,616	2,559
$\|E\|$	7,637	4,774	6,354	7,076	8,304	6,668
# groud truth		697		874		898
k	20	40	20	40	20	40
Random	0.0198	0.0566	0.0223	0.0447	0.0221	0.0521
RolX	0.0779	0.1288	0.0548	0.0984	0.0776	0.1309
Panther++	0.0892	0.1558	**0.0782**	0.1185	0.0921	0.1320
GraphWave	0.0846	**0.1693**	0.0549	0.0995	**0.0947**	**0.1470**
GCC (E2E)	**0.1047**	0.1564	0.0549	**0.1247**	0.0835	0.1336
GCC (MoCo)	0.0904	0.1521	0.0652	0.1178	0.0846	0.1425

GNN Pre-Training



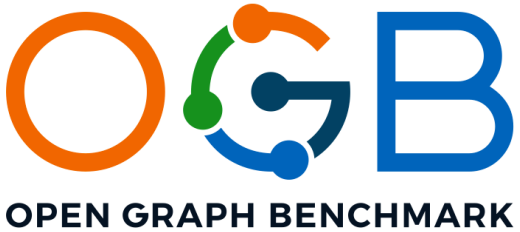
1. Jiezhong Qiu et al. GCC: Graph Contrastive Coding for Graph Neural Network Pre-Training. **KDD** 2020.
2. Code & Data for GCC: https://github.com/THUDM/GCC

GRL: NE&GNN

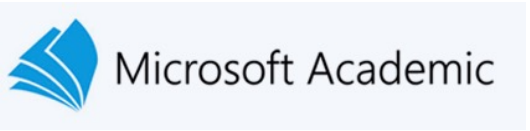
Network Embedding ⇔ Matrix Factorization

Graph Neural Networks ⇨ GNN Pre-Training

Learning GNNs with **CogDL**



https://alchemy.tencent.com/



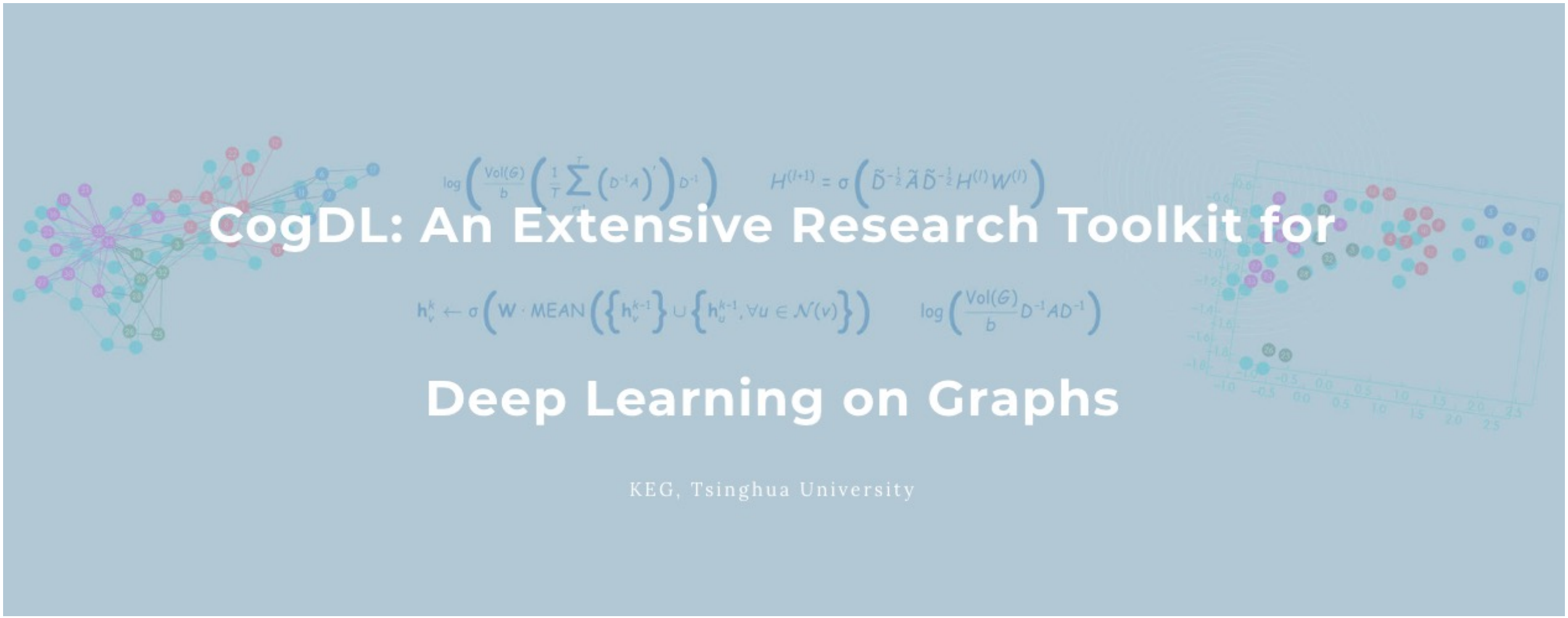
CogDL

https://github.com/thudm/cogdl

It is hard to apply GNNs to real-world applications…

- Billion-scale real-world graphs!
  - How to store large-scale graphs, GPU memory bounded
  - Large cost of model training
- Potential issues in training GNNs
  - Over-fitting, Over-smoothing
- Methodologies
  - “Millions of” papers…
  - Which one really works? Reproducibility…

CogDL


CogDL: An Extensive Research Toolkit for
Deep Learning on Graphs
KEG, Tsinghua University

CogDL Introduction

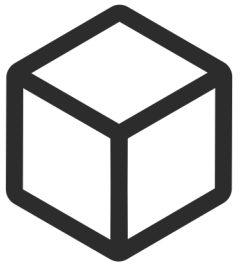
Vision

CogDL aims at providing researchers and developers with easy-to-use APIs, reproducible results, and high efficiency for most graph tasks and applications.

Philosophy

Easy-to-use

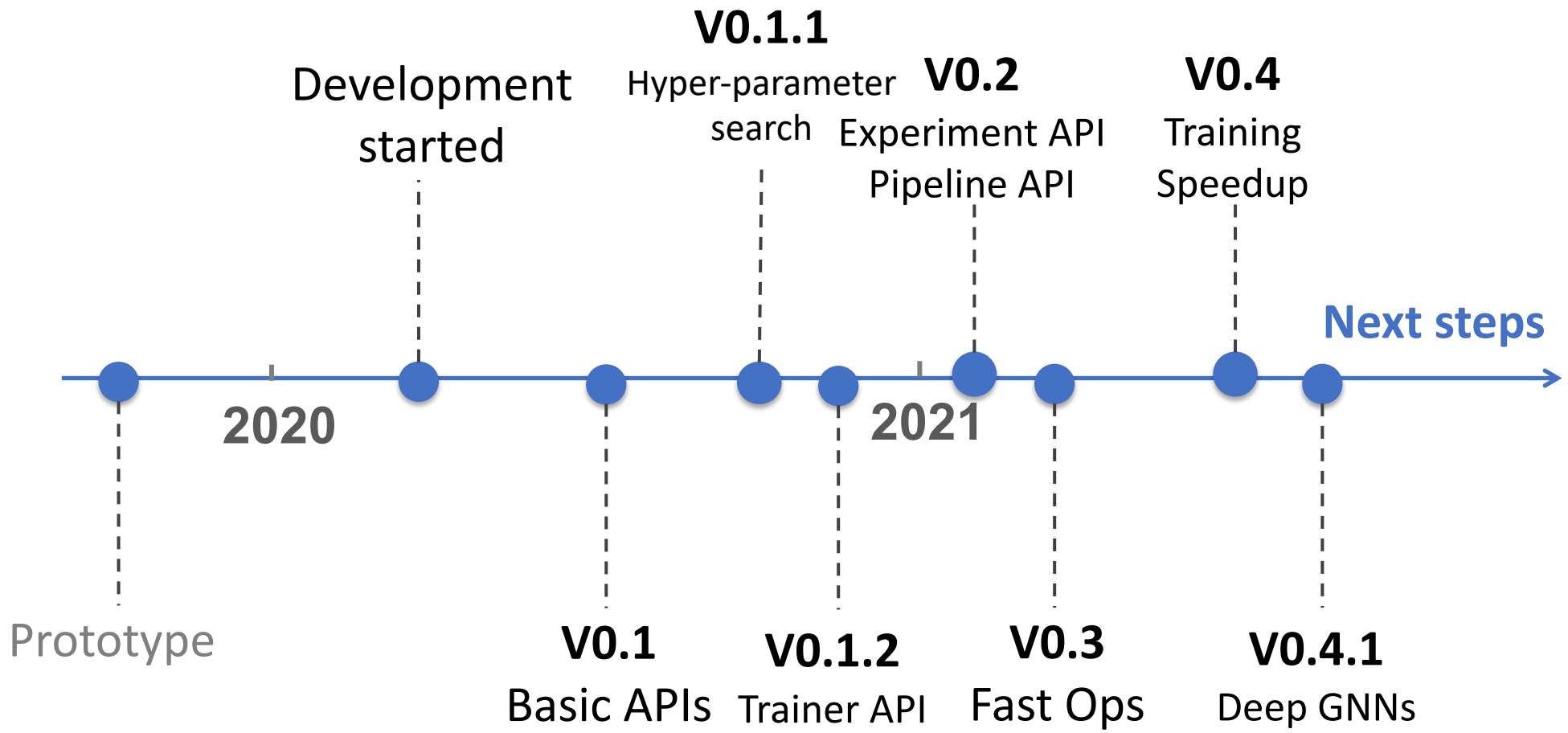
Reproducibility



Efficiency

1. Y. Cen, Z. Hou, Y. Wang, Q. Chen, Y. Luo, X. Yao, A. Zeng, S. Guo, Y. Yang, P. Zhang, G. Dai, Y. Wang, C. Zhou, H. Yang, and J. Tang. CogDL: An Extensive Toolkit for Deep Learning on Graphs. arXiv 2021.

CogDL Development

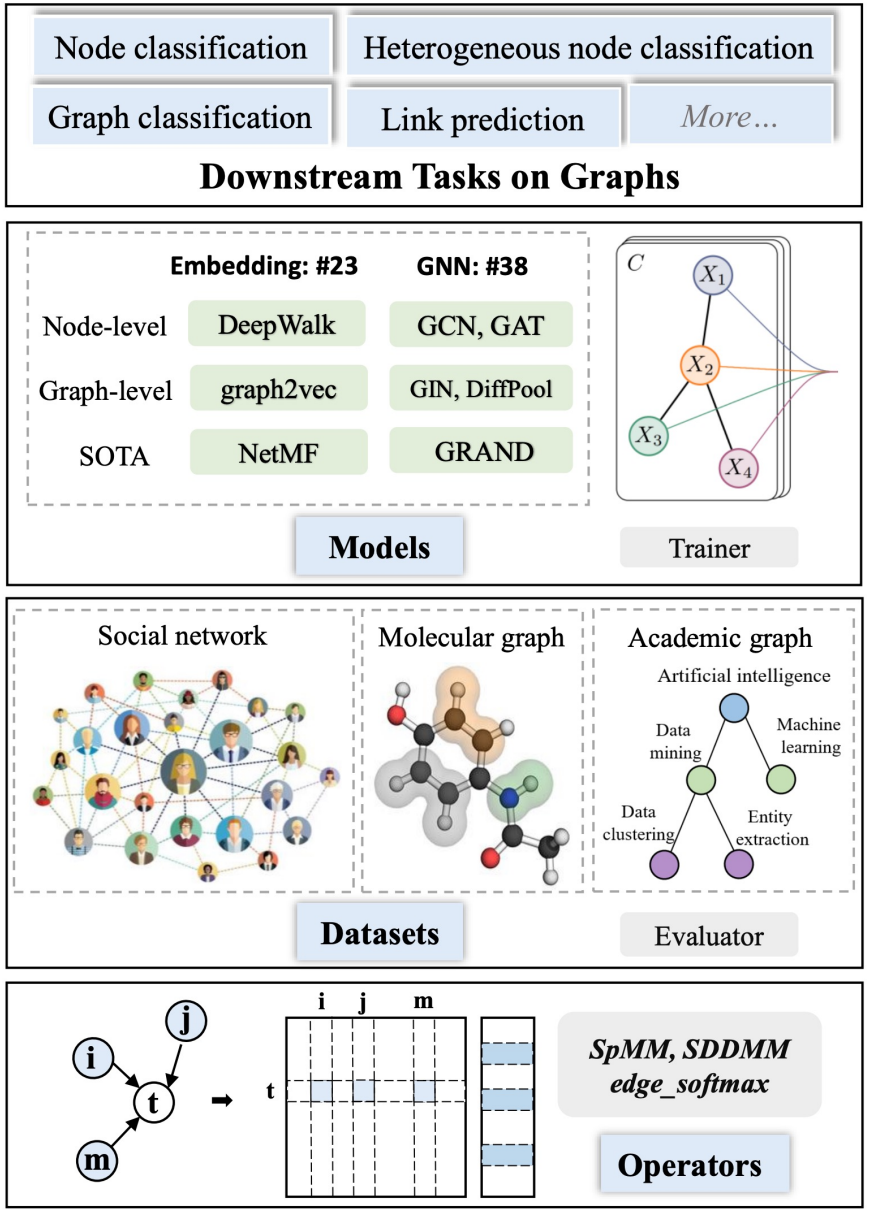


Prerequisite: PyTorch environment CogDL installation: **pip install cogdl**
or git clone https://github.com/THUDM/cogdl

Tasks, Datasets, Models in CogDL

- >10 Tasks
  - Node/graph classification, etc.
- >60 Datasets
  - Social networks, academics, molecular
- >70 models

Task	Characteristics		Models
Node Classification (Unsupervised)	*MF based*		SpectralClustering [7]
		With high-order neighborhood	NetMF [1], ProNE [2], NetSMF [3], HOPE [8], GraRep [9]
	Skip-gram		LINE [5]
		With high-order neighborhood	DeepWalk [6], Node2vec [4]
Node Classification (with GNN)	*Semi-supervised*		GCN [18], GAT [16], JK-Net [20], ChebyNet [21], GCNII [11]
		With random propagation	DropEdge [17], Graph-Unet [14], GraphSAINT [39], GRAND [10]
		With diffusion	GDC [15], APPNP [13], GRAND [10], PPRGo [40]
	Self-supervised	*Contrastive methods*	MVGRL [12], DGI [19]
Graph Classification	*Supervised*	*CNN method*	PATCHY_SAN [41]
		Global pooling	GIN [42], SortPool [43], DGCNN [44]
		Hierarchical pooling	DiffPool [45], SAGPool [46]
	Unsupervised	*Kernel methods*	DGK [47], graph2vec [48]
		GNN method	Infograph [49]



Experiment API

- Feed task, dataset, model, (hyper-parameters), (search space)

```python
from cogdl import experiment

# basic usage
experiment(task="node_classification", dataset="cora", model="gcn")

# set other hyper-parameters
experiment(task="node_classification", dataset="cora", model="gcn", hidden_size=32, max_epoch=200)

# run over multiple models on different seeds
experiment(task="node_classification", dataset="cora", model=["gcn", "gat"], seed=[1, 2])

# automl usage
def func_search(trial):
    return {
        "lr": trial.suggest_categorical("lr", [1e-3, 5e-3, 1e-2]),
        "hidden_size": trial.suggest_categorical("hidden_size", [32, 64, 128]),
        "dropout": trial.suggest_uniform("dropout", 0.5, 0.8),
    }

experiment(task="node_classification", dataset="cora", model="gcn", seed=[1, 2], func_search=func_search)
```

Results of Experiment API

In [3]:
```
from cogdl import experiment

# basic usage
experiment(task="node_classification", dataset="cora", model="gcn")
```

```
Failed to load C version of sampling, use python version instead.
Namespace(activation='relu', checkpoint=None, cpu=False, dataset='cora', device_id=[0], dropout=0.5, fast_spmm=False, hidden_s
ize=64, inference=False, lr=0.01, max_epoch=500, missing_rate=0, model='gcn', norm=None, num_classes=None, num_features=None,
num_layers=2, patience=100, residual=False, save_dir='.', save_model=None, seed=1, task='node_classification', trainer=None, u
se_best_config=False, weight_decay=0.0005)
Downloading https://cloud.tsinghua.edu.cn/d/6808093f7f8042bfa1f0/files/?p=%2Fcora.zip&dl=1
unpacking cora.zip
Processing...
Done!
Epoch: 455, Train: 1.0000, Val: 0.7920, ValLoss: 0.7300:  89%|████████▉ | 446/500 [00:03<00:00, 120.41it/s]
Valid accurracy =  0.7940
Test accuracy = 0.8100
| Variant          | Acc            | ValAcc         |
|------------------|----------------|----------------|
| ('cora', 'gcn') | 0.8100±0.0000 | 0.7940±0.0000 |
```

Out[3]:
```
defaultdict(list, {('cora', 'gcn'): [{'Acc': 0.81, 'ValAcc': 0.794}]})
```

Results of Node Classification

- Two kinds of models：
  - Semi-supervised：GCN, GAT, GRAND, …
  - Self-supervised：MVGRL, DGI
- Citation networks：Cora, Citeseer, Pubmed

Rank	Method	Cora	Citeseer	Pubmed	*Reproducible*
1	GRAND [12]	84.8	75.1	82.4	*Yes*
2	GCNII [7]	85.1	71.3	80.2	*Yes*
3	MVGRL [20]	83.6 ↓	73.0	80.1	*Partial*
4	APPNP [26]	**84.3** ↑	72.0	80.0	*Yes*
5	Graph-Unet [15]	83.3 ↓	71.2 ↓	79.0	*Partial*
6	GDC [27]	82.5	72.1	79.8	*Yes*
7	GAT [53]	82.9	71.0	78.9	*Yes*
8	DropEdge [38]	82.1	72.1	79.7	*Yes*
9	GCN [25]	**82.3** ↑	**71.4** ↑	79.5	*Yes*
10	DGI [52]	82.0	71.2	76.5	*Yes*
11	JK-net [58]	81.8	69.5	77.7	*Yes*
12	Chebyshev [8]	79.0	69.8	68.6	*Yes*

Results of Graph Classification

- Two kinds of models
  - Self-supervised：InfoGraph, graph2vec, DGK
  - Supervised：GIN, DiffPool, SortPool, …
- Two types of graphs
  - Bioinformatics：MUTAG, PTC, NCI1, PROTEINS
  - Social networks：IMDB-B/M, COLLAB, REDDIT-B

Algorithm	MUTAG	PTC	NCI1	PROTEINS	IMDB-B	IMDB-M	COLLAB	REDDIT-B	*Reproducible*
GIN [57]	92.06	67.82	81.66	75.19	76.10	51.80	79.52	83.10 ↓	*Yes*
InfoGraph [42]	88.95	60.74	76.64	73.93	74.50	51.33	79.40	76.55	*Yes*
DiffPool [62]	85.18	58.00	69.09	75.30	72.50	50.50	79.27	81.20	*Yes*
SortPool [67]	87.25	62.04	**73.99** ↑	74.48	75.40	50.47	**80.07** ↑	78.15	*Yes*
graph2vec [31]	83.68	54.76 ↓	71.85	73.30	73.90	52.27	**85.58** ↑	91.77	*Yes*
PATCHY_SAN [32]	86.12	61.60	69.82	75.38	**76.00** ↑	46.40	74.34	60.61	*Yes*
DGCNN [56]	83.33	56.72	65.96	66.75	71.60	49.20	77.45	86.20	*Yes*
SAGPool [28]	71.73 ↓	59.92	72.87	74.03	74.80	51.33	/	89.21	*Yes*
DGK [59]	85.58	57.28	/	72.59	55.00 ↓	40.40 ↓	/	/	*Partial*

Pipeline API

- Feed application, model, (hyper-parameters)

```python
import numpy as np
from cogdl import pipeline

# build a pipeline for generating embeddings
# pass model name with its hyper-parameters to this API
generator = pipeline("generate-emb", model="prone")

# generate embedding by an unweighted graph
edge_index = np.array([[0, 1], [0, 2], [0, 3], [1, 2], [2, 3]])
outputs = generator(edge_index)
print(outputs)

# build a pipeline for generating embeddings using unsupervised GNNs
# pass model name and num_features with its hyper-parameters to this API
generator = pipeline("generate-emb", model="dgi", num_features=8, hidden_size=4)
outputs = generator(edge_index, x=np.random.randn(4, 8))
print(outputs)
```

```
- prone
- netmf
- netsmf
- deepwalk
- line
- node2vec
- hope
- sdne
- grarep
- dngr
- spectral
```

```
- unsup_graphsage
- dgi
- mvgrl
- grace
```

All with CogDL

- Efficiency
  - Graph storage in CogDL
  - Sparse operators in CogDL
  - Training on large-scale graphs
  - Training very deep GNNs
- Customization
  - Customized usage in CogDL
- Benchmarks:
  - Self-supervised learning
  - Heterogeneous Graph Benckmark (HGB)
  - Graph Robustness Benchmark (GRB)
- Applications:
  - Recommendation

Sparse Storage of Adjacency Matrix

- COO format：
  - (row, col) or (row, col, value)，size: |E|*2/3
  - [[0,0,1], [0,2,2], [1,2,3], [2,0,4], [2,1,5], [2,2,6]]
- CSR format：
  - row_ptr：size |V|+1
  - col_indices: size |E|
  - value: size |E|
  - [0, 2, 3, 6], [0, 2, 2, 0, 1, 2], [1, 2, 3, 4, 5, 6]

$$\begin{bmatrix} 1 & 0 & 2 \\ 0 & 0 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

Graph Storage in CogDL

class Graph: (defined in cogdl.data)

- x：node feature matrix
- y：node labels
- edge_index：COO format matrix
- edge_weight：edge weight (if exists)
- edge_attr：edge attributes (if exists)
- row_ptr：row index pointer for CSR matrix
- col_indices：column indices for CSR matrix

Usage of CogDL's Graph

- Graph Initialization
  - g = Graph(edge_index=edge_index)
  - g.edge_weight = torch.rand(n)
- Commonly used operators：
  - add_self_loops()
  - sym_norm()
  - degrees()
  - subgraph()
  - …

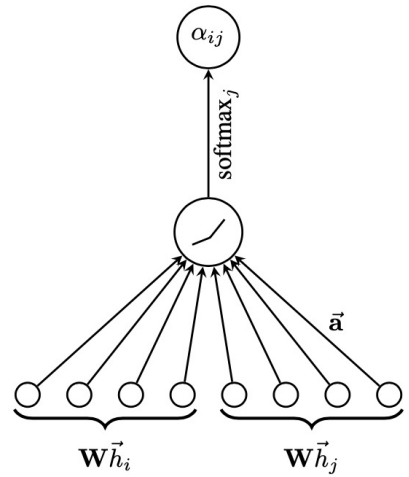
Recall Sparse Operators in GNNs

- GCN（Sparse Maxtrix-Matrix Multiplication, SpMM）

$$\boldsymbol{H}^{(i+1)} = \boldsymbol{A}\boldsymbol{H}^{(i)}\boldsymbol{W}$$

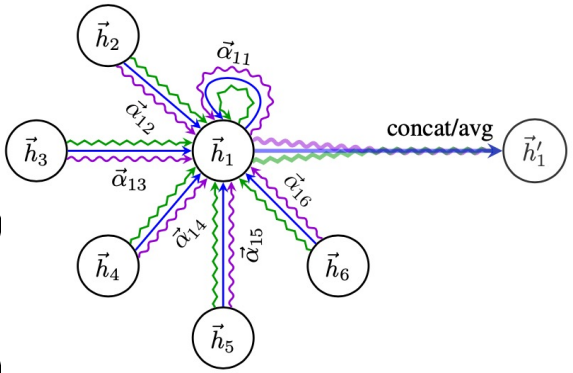
- GAT（ Edge-wise-softmax）

$$\alpha_{ij} = softmax(e_{ij}) = \frac{\exp(e_{ij})}{\sum_{k \in N_i} \exp(e_{ik})}$$



- GAT（Multi-Head SpMM）

$$\boldsymbol{h}_i = CONCAT\left(\sigma\left(\sum_{j \in N_i} \alpha_{ij}^k \boldsymbol{W}^k \boldsymbol{h}_j\right)\right)$$



GCN/GAT Layer in CogDL

```python
# GCN Layer
# graph: cogdl.data.Graph
# x: node featurs
# weight: parameters

h = torch.mm(x, weight)
h = spmm(graph, h)
out = torch.relu(h)
```

```python
# GAT Layer
# graph: cogdl.data.Graph
# h: node featurs
# h_score: importance score of edge

edge_attention = mul_edge_softmax(graph, edge_score)
h = mh_spmm(graph, edge_attention, h)
out = torch.cat(h, dim=1)
```

$$H^{(i+1)} = AH^{(i)}\boldsymbol{W}^{(i)}$$

$$\alpha_{ij} = softmax(e_{ij}) = \frac{\exp(e_{ij})}{\sum_{k\in N_i}\exp(e_{ik})}$$

$$h_i = CONCAT\left(\sigma\left(\sum_{j\in N_i}\alpha_{ij}^k \boldsymbol{W}^k h_j\right)\right)$$

Implementation of GCN/GAT Layer

```python
class GCNLayer(nn.Module):
    """
    Simple GCN layer, similar to https://arxiv.o
    """

    def __init__(self, in_features, out_features

    def reset_parameters(self):…

    def forward(self, graph, x):
        support = torch.mm(x, self.weight)
        out = spmm(graph, support)
```

```python
class GATLayer(nn.Module):
    """
    Sparse version GAT layer, similar to https://arxiv.org/ab
    """

    def __init__(self, in_features, out_features, nhead=1, al

    def reset_parameters(self):…

    def forward(self, graph, x):
        h = torch.matmul(x, self.W)
        edge_score = self.compute_edge_score(graph, x, h)
        # edge_attention: E * H
        edge_attention = mul_edge_softmax(graph, edge_score)
        edge_attention = self.dropout(edge_attention)

        out = mh_spmm(graph, edge_attention, h)
```

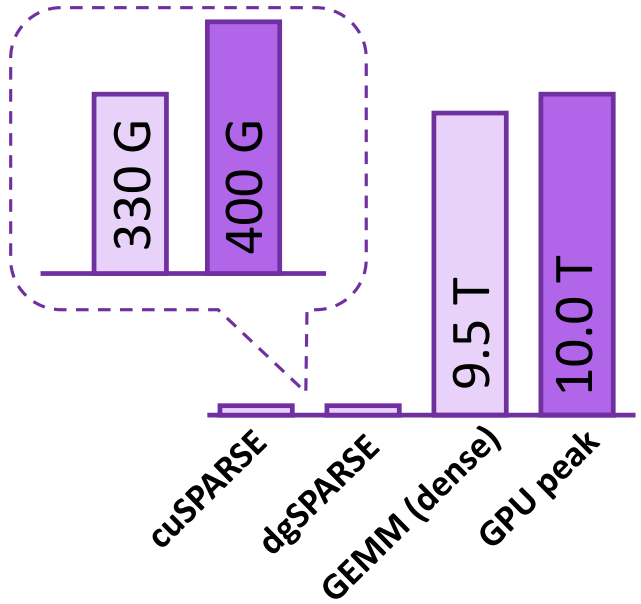
$$H^{(i+1)} = AH^{(i)}\boldsymbol{W}^{(\mathrm{i})}$$

$$\alpha_{ij} = softmax(e_{ij}) = \frac{\exp(e_{ij})}{\sum_{k \in N_i} \exp(e_{ik})}$$

$$h_i = CONCAT\left(\sigma\left(\sum_{j \in N_i} \alpha_{ij}^k \boldsymbol{W}^k h_j\right)\right)$$

Efficient Sparse Kernels

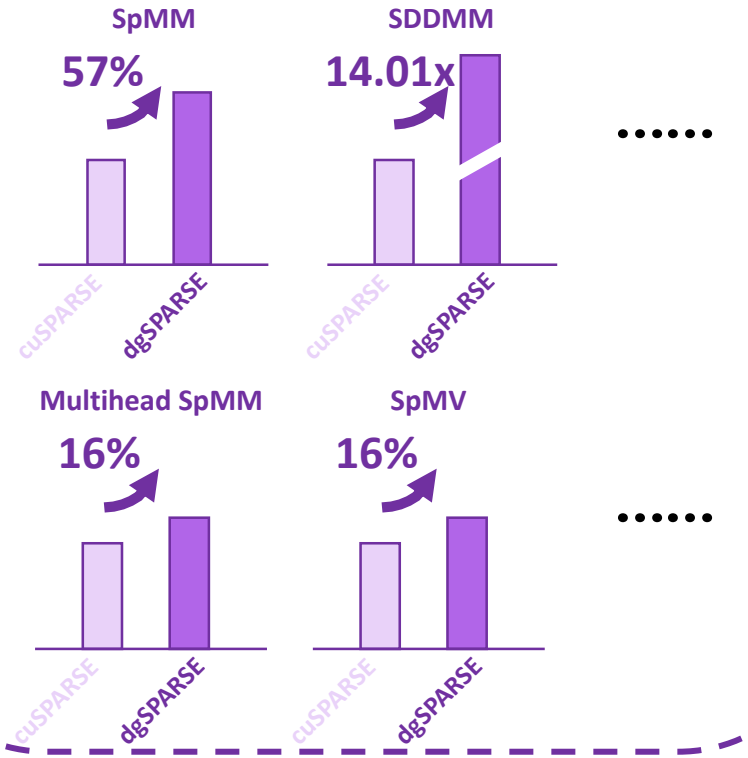
Lack efficient sparse kernels



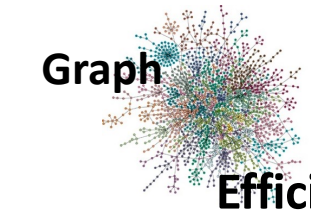
Big gap between FLOPS of sparse kernels and hardware peak performance.

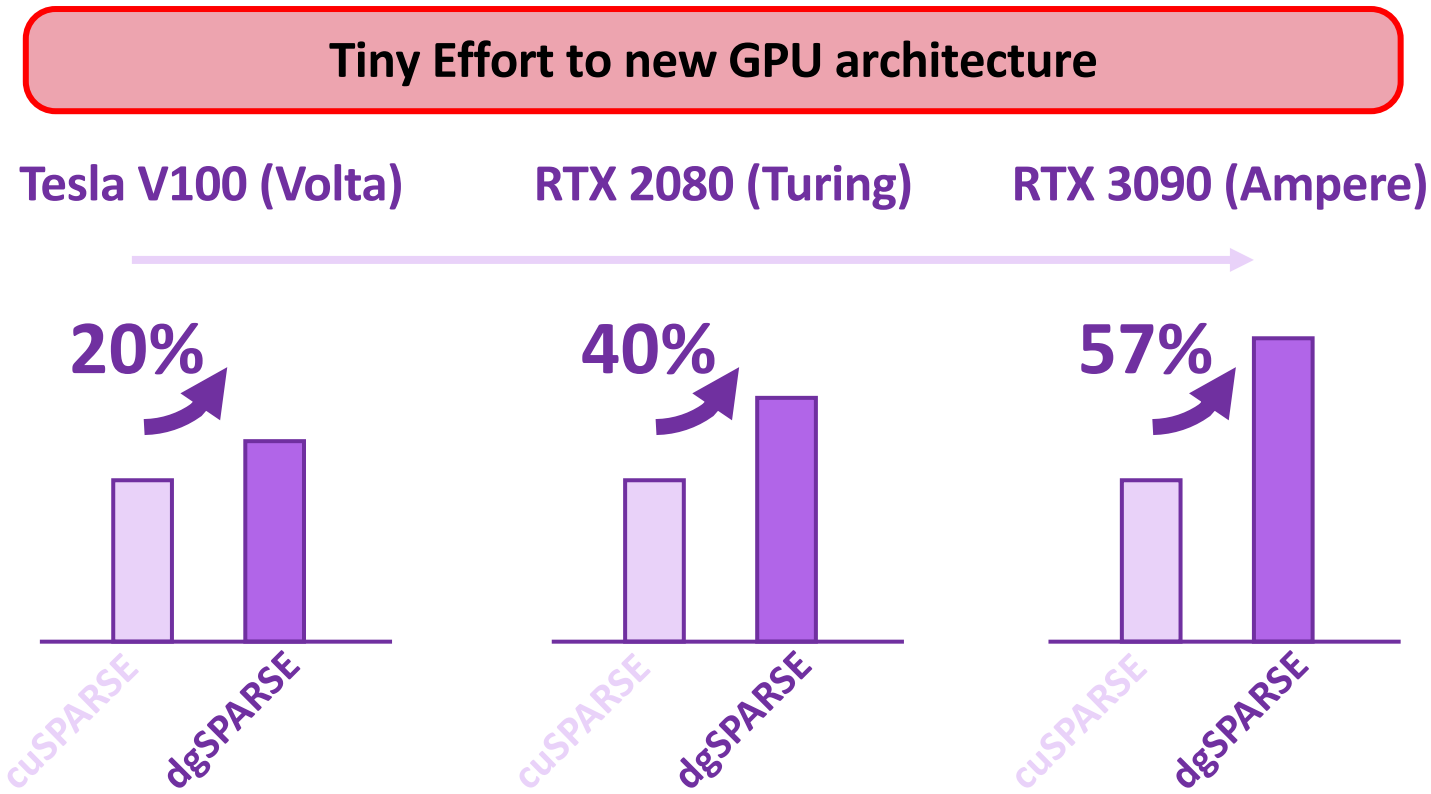
dgSPARSE, Deep Graph SPARSE

Efficient Implementation on GPUs



Deep Graph Sparse (dgSPARSE) Library





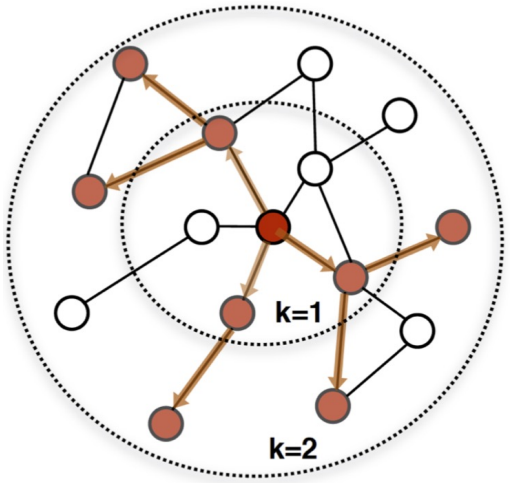
Performance of GCN/GAT model

- Setting：2-layer GCN/GAT，hidden size=128
- Supported by dgSPAESE

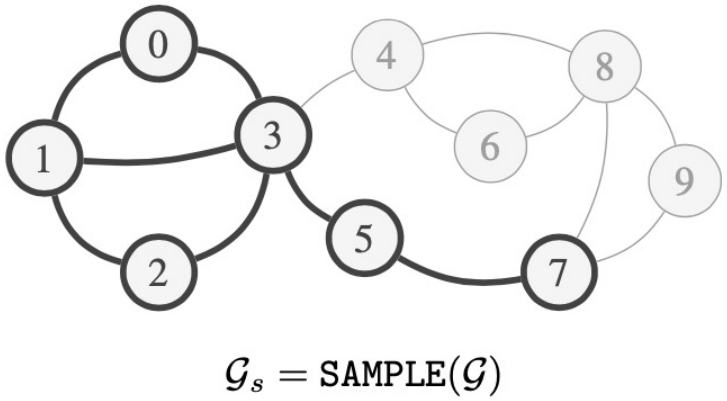
Model	GPU	Dataset	Tranining per epoch (s)				Inference per epoch (s)			
			torch	PyG	DGL	CogDL	torch	PyG	DGL	CogDL
GCN	2080Ti (11G)	Flickr	0.025	0.0084	0.012	0.085	0.012	0.0034	0.007	0.0035
		Reddit	0.445	0.122	0.102	0.081	0.218	0.045	0.049	0.039
		Yelp	0.412	0.151	0.151	0.110	0.191	0.053	0.063	0.040
	3090 (24G)	Flickr	0.017	0.006	0.008	0.007	0.008	0.002	0.004	0.002
		Reddit	0.263	0.062	0.060	0.050	0.127	0.022	0.0314	0.022
		Yelp	0.230	0.081	0.080	0.062	0.106	0.029	0.036	0.023
GAT	2080Ti (11G)	PubMed	0.017	0.016	0.011	0.012	0.004	0.006	0.004	0.003
		Flickr	0.082	0.090	0.047	0.056	0.023	0.030	0.019	0.014
		⋆Reddit	__‡	__‡	0.406	0.537	__‡	__‡	0.163	0.086
	3090 (24G)	PubMed	0.043	0.011	0.011	0.016	0.004	0.004	0.003	0.002
		Flickr	0.097	0.059	0.033	0.044	0.021	0.024	0.013	0.009
		Reddit	0.671	__‡	0.301	0.373	0.112	__‡	0.113	0.088
		Yelp	0.614	__‡	0.294	0.404	0.118	__‡	0.105	0.113

Training on Large-scale Graphs

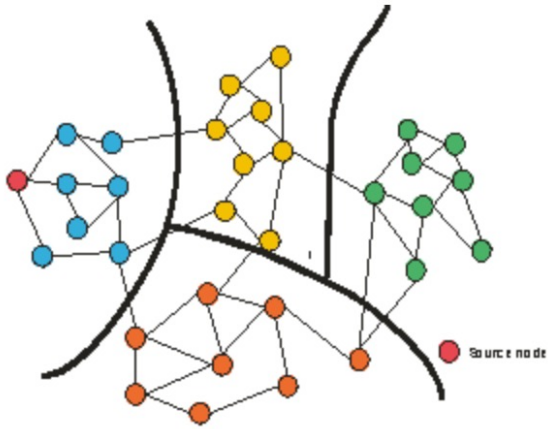
- Billion-scale social networks and recommender systems
- Main challenge: GPU memory bounded!
- Training GNNs via mini-batch sampling



Neighbor Sampling
(NeurIPS ’17)



GraphSAINT
(ICLR ’20)

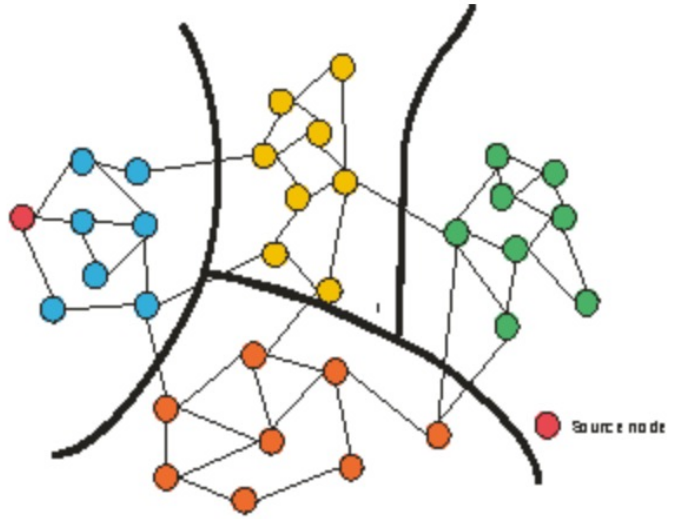


ClusterGCN
(KDD ’19)

1. Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. NeurIPS ’17.
2. Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. Cluster-GCN: An efficient algorithm for training deep and large graph convolutional networks. KDD ’19.
3. Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. Graphsaint: Graph sampling based inductive learning method. In ICLR ’20.

ClusterGCN

- Graph partition (METIS)
- Train GNNs via mini-batch



- Memory: O(NFL) -> O(bFL)

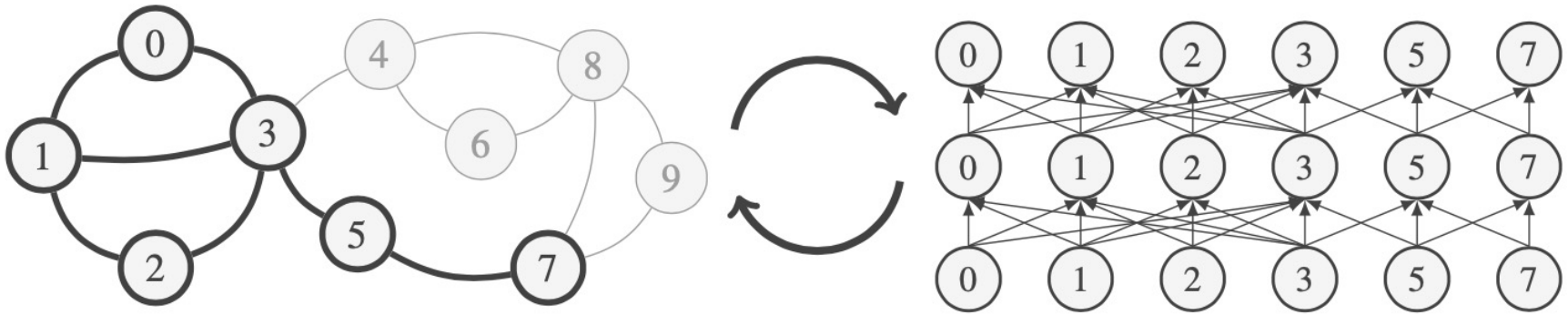
	GCN [9]	Vanilla SGD	GraphSAGE [5]	FastGCN [1]	VR-GCN [2]	Cluster-GCN
Time complexity	$O(L\|\|A\|\|_0F + LNF^2)$	$O(d^LNF^2)$	$O(r^LNF^2)$	$O(rLNF^2)$	$O(L\|\|A\|\|_0F + LNF^2 + r^LNF^2)$	$O(L\|\|A\|\|_0F + LNF^2)$
Memory complexity	$O(LNF + LF^2)$	$O(bd^LF + LF^2)$	$O(br^LF + LF^2)$	$O(brLF + LF^2)$	$O(LNF + LF^2)$	$O(bLF + LF^2)$

	Time		Memory		Test F1 score	
	VRGCN	Cluster-GCN	VRGCN	Cluster-GCN	VRGCN	Cluster-GCN
Amazon2M (2-layer)	337s	1223s	7476 MB	2228 MB	89.03	89.00
Amazon2M (3-layer)	1961s	1523s	11218 MB	2235 MB	90.21	90.21
Amazon2M (4-layer)	N/A	2289s	OOM	2241 MB	N/A	90.41

GraphSAINT

- Unbiased sampler
  - Random node/edge/walk sampler
- Unbiased aggregated representations

$$\zeta_v^{(\ell+1)} = \sum_{u \in \mathcal{V}} \frac{\widetilde{\boldsymbol{A}}_{v,u}}{\alpha_{u,v}} \left(\boldsymbol{W}^{(\ell)}\right)^{\mathsf{T}} \boldsymbol{x}_u^{(\ell)} \mathbb{1}_{u|v} = \sum_{u \in \mathcal{V}} \frac{\widetilde{\boldsymbol{A}}_{v,u}}{\alpha_{u,v}} \tilde{\boldsymbol{x}}_u^{(\ell)} \mathbb{1}_{u|v}$$



$\mathcal{G}_s = \texttt{SAMPLE}(\mathcal{G})$

Full GCN on $\mathcal{G}_s$

GraphSAINT Performance

Dataset	Nodes	Edges	Degree	Feature	Classes	Train / Val / Test
PPI	14,755	225,270	15	50	121 (m)	0.66 / 0.12 / 0.22
Flickr	89,250	899,756	10	500	7 (s)	0.50 / 0.25 / 0.25
Reddit	232,965	11,606,919	50	602	41 (s)	0.66 / 0.10 / 0.24
Yelp	716,847	6,977,410	10	300	100 (m)	0.75 / 0.10 / 0.15
Amazon	1,598,960	132,169,734	83	200	107 (m)	0.85 / 0.05 / 0.10
PPI (large version)	56,944	818,716	14	50	121 (m)	0.79 / 0.11 / 0.10

Method	PPI	Flickr	Reddit	Yelp	Amazon
GCN	0.515±0.006	0.492±0.003	0.933±0.000	0.378±0.001	0.281±0.005
GraphSAGE	0.637±0.006	0.501±0.013	0.953±0.001	0.634±0.006	0.758±0.002
FastGCN	0.513±0.032	0.504±0.001	0.924±0.001	0.265±0.053	0.174±0.021
S-GCN	0.963±0.010	0.482±0.003	0.964±0.001	0.640±0.002	—‡
AS-GCN	0.687±0.012	0.504±0.002	0.958±0.001	—‡	—‡
ClusterGCN	0.875±0.004	0.481±0.005	0.954±0.001	0.609±0.005	0.759±0.008
GraphSAINT-Node	0.960±0.001	0.507±0.001	0.962±0.001	0.641±0.000	0.782±0.004
GraphSAINT-Edge	**0.981**±0.007	0.510±0.002	**0.966**±0.001	**0.653**±0.003	0.807±0.001
GraphSAINT-RW	**0.981**±0.004	**0.511**±0.001	**0.966**±0.001	**0.653**±0.003	**0.815**±0.001
GraphSAINT-MRW	0.980±0.006	0.510±0.001	0.964±0.000	0.652±0.001	0.809±0.001

Multi-GPU Training

Multi-GPU implementation:

sampling + PyTorch DDP

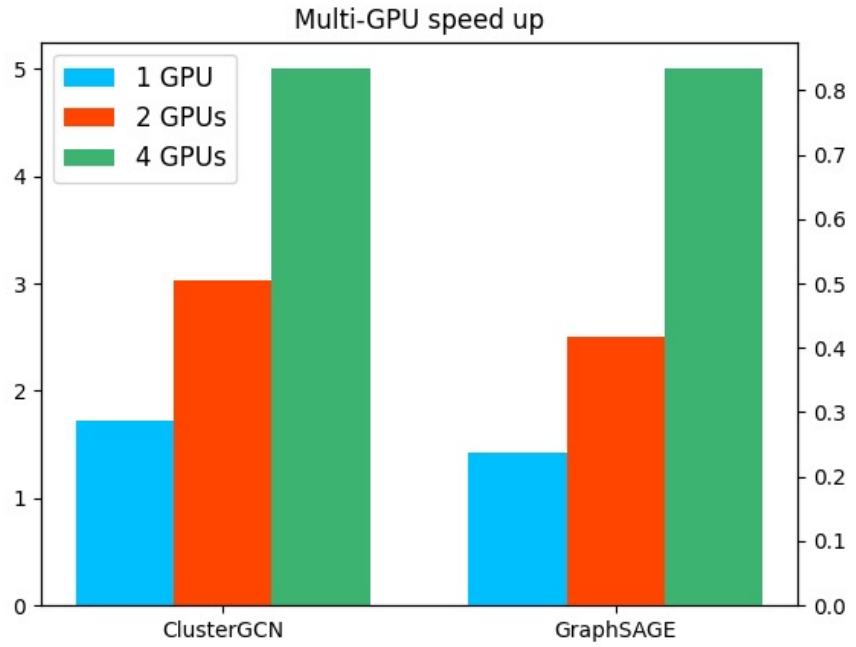
✓ Sampler in CogDL

[+] NeighborSampling

[+] ClusterGCN

[+] GraphSAINT

✓ 4 GPUs ~ 3x↑ speedup



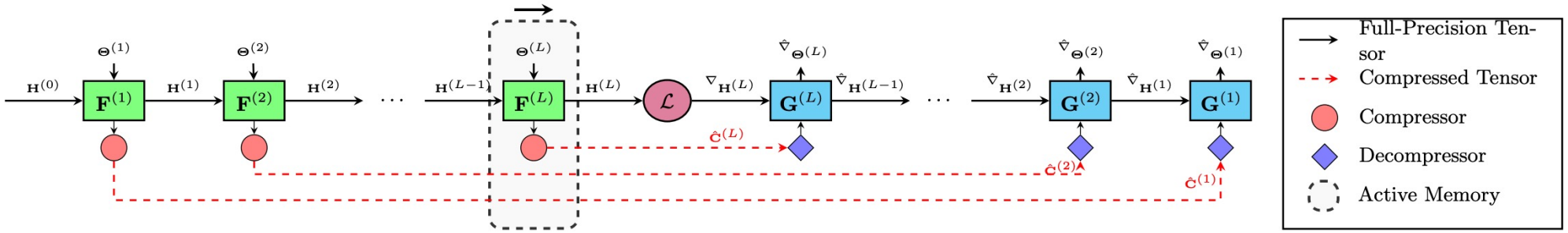
Usage: python scripts/train.py --model gcn --task node_classification --dataset reddit --trainer dist_clustergcn

Other Solutions for very Deep GNNs?

- GPU memory is the bottleneck for training very deep GNNs.
- Recall RevGNN uses reversible blocks.
- Are there other solutions?
- Activation Compressed Training!

ActNN : Activation Compressed Training

- ActNN : Reducing Training Memory Footprint via **2-Bit Activation** Compressed Training (By Jianfei Chen, Tsinghua)
- “ActNN reduces the memory footprint of the activation by 12×.”
- https://github.com/ucbrise/actnn



1. Chen, Jianfei, et al. "ActNN: Reducing Training Memory Footprint via 2-Bit Activation Compressed Training." *arXiv preprint arXiv:2104.14129* (2021).

ActNN Theory

Theorem 1. *(Unbiased Gradient) There exists random quantization strategies for* $\hat{\mathbf{C}}$*, such that*

$$\mathbb{E}\left[\hat{\nabla}_{\mathbf{\Theta}}\right] = \nabla_{\mathbf{\Theta}} \mathcal{L}_{\mathcal{D}}(\mathbf{\Theta}).$$

Theorem 2. *(Convergence) If A1-A3 holds, and* $0 < \alpha \leq \frac{1}{\beta}$*, take the number of iterations* t *uniformly from* $\{1, \ldots, T\}$*, where* T *is a maximum number of iterations. Then*

$$\mathbb{E}\left\|\nabla \mathcal{L}_{\mathcal{D}}(\mathbf{\Theta}_t)\right\|^2 \leq \frac{2(\mathcal{L}(\Theta_1) - \mathcal{L}_{inf})}{\alpha T} + \alpha\beta\sigma^2. \tag{6}$$

Theorem 3. *(Gradient Variance)*

$$\mathrm{Var}\left[\hat{\nabla}_{\mathbf{\Theta}^{(l)}}\right] = \mathrm{Var}\left[\nabla_{\mathbf{\Theta}^{(l)}}\right] + \sum_{m=l}^{L} \mathbb{E}\left[\mathrm{Var}\left[\mathbf{G}_{\mathbf{\Theta}}^{(l\sim m)}\left(\hat{\nabla}_{\mathbf{H}^{(m)}}, \hat{\mathbf{C}}^{(m)}\right) \,\middle|\, \hat{\nabla}_{\mathbf{H}^{(m)}}\right]\right].$$

ActNN Implementation

```
class RegularLayer:
  def forward(context, input):
    context.save_for_backward(input)
    return compute_output(input)

  def backward(context, grad_output):
    input = context.saved_tensors
    return compute_gradient(grad_output, input)

class ActivationCompressedLayer:
  def forward(context, input):
    context.save_for_backward(compress(input))
    return compute_output(input)

  def backward(context, grad_output):
    input = decompress(context.saved_tensors))
    return compute_gradient(grad_output, input)
```

ActNN Performance

- Experiment on ImageNet

Bits	32	4	3	2	1.5	1.25
FP	**77.1**	N/A	N/A	N/A	N/A	N/A
BLPA	N/A	**76.6**	Div.	Div.	N/A	N/A
ActNN (L2)	N/A	-	**77.4**	0.1	N/A	N/A
ActNN (L2.5)	N/A	-	-	**77.1**	75.9	75.1
ActNN (L3)	N/A	-	-	**76.9**	76.4	75.9

Network	Batch	Total Mem. (GB)			Act. Mem. (GB)		
		FP	ActNN (L3)	R	FP	ActNN (L3)	R
ResNet-152	32	6.01	1.18	5×	5.28	0.44	12×
	64	11.32	1.64	7×	10.57	0.88	12×
	96	OOM	2.11	/	OOM	1.32	/
	512	OOM	8.27	/	OOM	7.01	/
FCN-HR-48	2	5.76	1.39	4×	4.76	0.39	12×
	4	10.52	1.79	6×	9.52	0.79	12×
	6	OOM	2.17	/	OOM	1.18	/
	20	OOM	4.91	/	OOM	3.91	/

When SpMM meets ActNN (in CogDL)

$$H^{(i+1)} = AH^{(i)}W$$

```python
class SPMMFunction(torch.autograd.Function):
    @staticmethod
    def forward(ctx, rowptr, colind, feat, edge_weight_csr=None, sym=False):
        if edge_weight_csr is None:
            out = spmm.csr_spmm_no_edge_value(rowptr, colind, feat)
        else:
            out = spmm.csr_spmm(rowptr, colind, edge_weight_csr, feat)
        ctx.backward_csc = (rowptr, colind, feat, edge_weight_csr, sym)
        return out

    @staticmethod
    def backward(ctx, grad_out):
        rowptr, colind, feat, edge_weight_csr, sym = ctx.backward_csc
        if edge_weight_csr is not None:
            grad_out = grad_out.contiguous()
            if sym:
                colptr, rowind, edge_weight_csc = rowptr, colind, edge_weigh
            else:
                colptr, rowind, edge_weight_csc = spmm.csr2csc(rowptr, colin
            grad_feat = spmm.csr_spmm(colptr, rowind, edge_weight_csc, grad_
            grad_edge_weight = sddmm.csr_sddmm(rowptr, colind, grad_out, fea
        else:
            if sym is False:
                colptr, rowind, edge_weight_csc = spmm.csr2csc(rowptr, colin
                grad_feat = spmm.csr_spmm_no_edge_value(colptr, rowind, grad
            else:
                grad_feat = spmm.csr_spmm_no_edge_value(rowptr, colind, grad
            grad_edge_weight = None
        return None, None, grad_feat, grad_edge_weight, None
```

```python
class ActSPMMFunction(torch.autograd.Function):
    @staticmethod
    def forward(ctx, rowptr, colind, feat, edge_weight_csr=None, sym=False):
        if edge_weight_csr is None:
            out = spmm.csr_spmm_no_edge_value(rowptr, colind, feat)
        else:
            out = spmm.csr_spmm(rowptr, colind, edge_weight_csr, feat)
        quantized = quantize_activation(feat, None)
        ctx.backward_csc = (rowptr, colind, quantized, edge_weight_csr, sym)
        ctx.other_args = feat.shape

        return out

    @staticmethod
    def backward(ctx, grad_out):
        rowptr, colind, quantized, edge_weight_csr, sym = ctx.backward_csc
        q_input_shape = ctx.other_args
        feat = dequantize_activation(quantized, q_input_shape)
        del quantized, ctx.backward_csc

        if edge_weight_csr is not None:
            grad_out = grad_out.contiguous()
            if sym:
                colptr, rowind, edge_weight_csc = rowptr, colind, edge_weight_cs
            else:
                colptr, rowind, edge_weight_csc = spmm.csr2csc(rowptr, colind, e
            grad_feat = spmm.csr_spmm(colptr, rowind, edge_weight_csc, grad_out)
            grad_edge_weight = sddmm.csr_sddmm(rowptr, colind, grad_out, feat)
```

Experimental Results

- Default setting of CogDL

Dataset	Origin GCN	GCN + actnn
Cora	81.30 ± 0.22	81.27 ± 0.19
Citeseer	71.73 ± 0.54	71.70 ± 0.28
Pubmed	79.17 ± 0.12	79.10 ± 0.08
Flickr	50.74 ± 0.10	50.89 ± 0.04
Reddit	95.01 ± 0.02	94.89 ± 0.01

Activation Memory (GCN + ActNN)

- Setting： $\boldsymbol{H} = \mathrm{Dropout}\left(\mathrm{ReLU}\left(\mathrm{BN}(\boldsymbol{AHW})\right)\right)$

#dataset, #layers, #hidden	Origin GCN	GCN + actnn	ratio	Ideal ratio
PPI, 5, 2048	3704	420	8.8x	raw: 32*2 / (2.125*2+1) =12.2x +bn: 32*3 / (2.125*3+1) = 13.0x +bn+dropout: 32*4 / (2.125*3+2)=15.3x
PPI, 5, 2048 (+bn)	5484	539	10.2x	
PPI, 5, 2048 (+bn, +dropout)	7711	594	13.0x	
Flickr, 5, 512	1420	154	9.2x	
Flickr, 5, 512 (+bn)	2117	201	10.5x	
Flickr, 5, 512 (+bn, +dropout)	2991	223	13.4x	
Flickr, 10, 512	3178	311	10.2x	
Flickr, 10, 512 (+bn)	4747	415	11.4x	
Flickr, 10, 512 (+bn, +dropout)	6712	465	14.4x	

Activation Memory (GraphSAGE + ActNN)

- Setting：$\boldsymbol{H} = \mathrm{Dropout}\left(\mathrm{ReLU}\left(\mathrm{BN}\left(\mathrm{Concat}\left(\boldsymbol{AH}, \boldsymbol{H}\right)\boldsymbol{W}\right)\right)\right)$

#dataset, #layers, #hidden	Origin SAGE	SAGE + actnn	ratio	Ideal ratio
PPI, 5, 2048	5524	580	9.5x	raw: 32*2 / (2.125*2+1) =12.2x +bn: 32*3 / (2.125*3+1) = 13.0x +bn+dropout: 32*4 / (2.125*3+2)=15.3x
PPI, 5, 2048 (+bn)	7304	698	10.5x	
PPI, 5, 2048 (+bn, +dropout)	OOM	754	-	
Flickr, 5, 512	2457	209	11.8x	
Flickr, 5, 512 (+bn)	3155	255	12.4x	
Flickr, 5, 512 (+bn, +dropout)	4027	278	14.5x	
Flickr, 10, 512	5090	430	11.8x	
Flickr, 10, 512 (+bn)	6659	534	12.5x	
Flickr, 10, 512 (+bn, +dropout)	8624	584	14.8x	

Activation Memory (GCNII + ActNN)

- Setting：

$$\mathbf{H}^{(\ell+1)}=\sigma\left(\left((1-\alpha_\ell)\tilde{\mathbf{P}}\mathbf{H}^{(\ell)}+\alpha_\ell\mathbf{H}^{(0)}\right)\left((1-\beta_\ell)\mathbf{I}_n+\beta_\ell\mathbf{W}^{(\ell)}\right)\right)$$

#dataset, #layers, #hidden	GCNII	GCNII + actnn	ratio	ideal
PPI, 10, 512	4008	340	11.8x	(32*3) / (2*2.125+2)=15.36
PPI, 20, 512	7708	619	12.5x	
PPI, 20, 1024	7879	603	13.1x	
Flickr, 5, 512	3421	229	14.9x	
Flickr, 10, 512	6268	413	15.2x	
Flickr, 5, 1024	6660	438	15.2x	

#layers	Origin GCNII	GCNII + actnn
32	84.83 ± 0.33	84.67 ± 0.12
64	85.13 ± 0.61	85.00 ± 0.37
128	84.83 ± 0.58	85.20 ± 0.36
256	85.00 ± 0.08	85.37 ± 0.54

Activation Memory (GIN + ActNN)

- Setting： $\boldsymbol{H}^{(l)} = \mathrm{MLP}^{(l)}\left((1+\epsilon)\boldsymbol{H}^{(l-1)} + \boldsymbol{A}\boldsymbol{H}^{(l-1)}\right)$

$$\boldsymbol{h}_G = \mathrm{CONCAT}(\mathrm{READOUT}(\boldsymbol{H}^{(l)}), l = 0, 1, \ldots L$$

#dataset, #batch, #layers, #hidden	GIN	GIN + actnn	ratio	ideal
NCI1, 512, 20, 512	2723	262	10.4x	(32*3) / (2.125*3+1) = 13.0x
NCI1, 512, 20, 1024	5735	540	10.6x	
NCI1, 1024, 20, 512	5502	528	10.4x	
NCI1, 512, 40, 512	6231	590	10.6x	

Self-supervised Learning on Graphs

- Types of self-supervision：
  - generative learning
  - contrastive learning
- Learning paradigm：
  - Pre-training & Fine-tuning
  - Joint learning
  - Self-training
- Encoders：GCN, GAT, GIN
- Downstream tasks

Survey of Graph Self-supervised Learning

Table 1: An overview of recent generative models on graphs. For acronyms used, "PT+FT" refers to Pre-training and Fine-tuning, "JL" refers to Joint Learning, "ST" refers to Self-Training.

Model	Self-Supervision Tasks	Training Scheme	Graph Encoders	Downstream Tasks
GAE/VGAE[1]	Graph Reconstruction	PT+FT	GCN	Node/Link
EdgeMask[2]	Graph Reconstruction	JL/PT+FT	GCN	Node
AttributeMask[2]	Graph Reconstruction	JL/PT+FT	GCN	Node
GraphCompletion[3]	Graph Reconstruction	JL/PT+FT	GCN/GAT/GIN/GraphMix	Node
GPT-GNN[4]	Graph Reconstruction	PT+FT	Heterogeneous Graph Transformer	Node/Link
S^2GRL[5]	Graph Property Prediction	PT+FT	GCN/GraphSAGE	Node
Distance2Cluster[2]	Graph Property Prediction	JL/PT+FT	GCN	Node
Node clustering[3]	Graph Property Prediction	JL/PT+FT	GCN/GAT/GIN/GraphMix	Node
Graph partitioning[3]	Graph Property Prediction	JL/PT+FT	GCN/GAT/GIN/GraphMix	Node
SuperGAT[6]	Graph Property Prediction	JL	SuperGAT	Node
Un_GraphSAGE[7]	Graph Property Prediction	PT+FT	GraphSAGE	Node
M3S[8]	Graph Property Prediction	ST	GCN	Node
AdvT[3]	Adversial Attack and Defense	JL	GCN	Node
Graph-Bert[9]	Hybrid	PT+FT	Graph Transformer	Node

Table 2: An overview of recent contrastive models on graphs.

Model	Augmentation Method	Contrastive Framework	Graph Encoders	Downstream Tasks
InfoGraph[10]	Original Graph	Deep Infomax	GIN	Graph
GraphCL[11]	Graph Corruption		GCN	Graph
GRACE[12]	Graph Corruption	InfoNCE	GCN	Node
GCC[13]	Random walk Sampling	MoCo	GIN	Node/Graph
DGI[14]	Graph Corruption + Sampling	Deep Infomax	GCN	Node
MVGRL[15]	Graph Corruption + Sampling	Deep Infomax	GCN/GraphSAGE	Node
DwGCL[16]	Graph Corruption + Sampling		GCN	Node

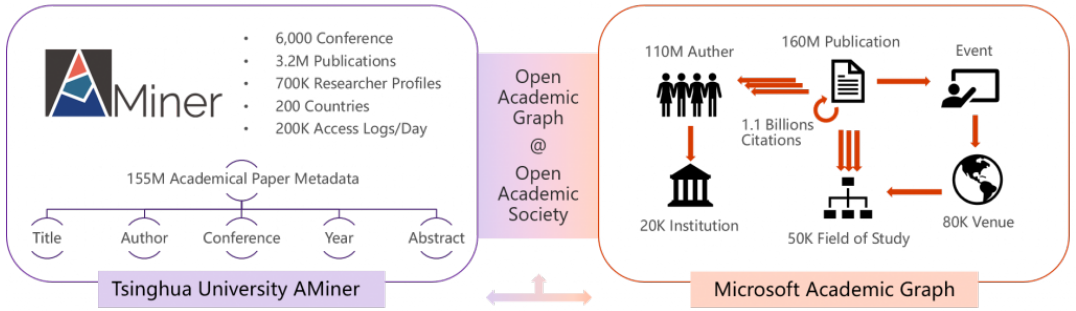
Results of Self-supervised Learning

- Learning paradigm:
  - Self-supervised (SL), Joint Learning (JL), unsupervised representation learning (URL)
- Semi-supervised datasets：Cora, Citeseer, PubMed
- Supervised datasets：Flickr, Reddit

Model	Training Scheme	Cora	Citeseer	PubMed	Flickr	Reddit
Linear	SL	47.86±0.02	49.25±0.04	69.17±0.03	45.81±0.00	67.95±0.01
GCN	SL	81.53± 0.26	71.75±0.05	79.30±0.31	52.74±0.13	95.16±0.01
EdgeMask	JL	81.28±0.31	71.53±0.24	79.55±0.11	52.55±0.06	95.07±0.01
AttributeMask	JL	81.20±0.32	71.45±0.35	78.93±0.25	52.45±0.15	95.00±0.01
S^2GRL	JL	83.42±0.63	72.37±0.11	81.20±0.37	52.59±0.05	95.17±0.00
Distance2Clusters	JL	82.47±0.48	71.55±0.30	81.53±0.22	52.24±0.07	/
SuperGAT	JL	82.77±0.53	72.25±0.52	80.30±0.31	52.80±0.07	95.42±0.01
MVGRL	URL	80.76±0.80	65.84±2.72	76.01±2.46	46.08±0.39	92.76±0.22
DGI	URL	81.91±0.17	70.01±0.87	76.49±1.04	46.35±0.11	93.12±0.18
EdgeMask	URL	75.23±1.14	68.96±0.96	79.41±1.15	50.48±0.06	93.68±0.01
AttributeMask	URL	76.12±0.91	70.69±0.44	75.16±1.71	51.57±0.15	93.37±0.01
S^2GRL	URL	81.56±0.49	69.48±0.91	80.83±0.57	50.85±0.03	93.88±0.05
Distance2Cluster	URL	73.86±0.29	66.53±0.29	79.44±0.34	50.24±0.07	/

OAG: Open Academic Graph

https://www.openacademic.ai/oag/



Data set	#Pairs/Venues	Date
Linking relations	29,841	2018.12
AMiner venues	69,397	2018.07
MAG venues	52,678	2018.11

Table 1: statistics of OAG venue data

Data set	#Pairs/Papers	Date
Linking relations	91,137,597	2018.12
AMiner papers	172,209,563	2019.01
MAG papers	208,915,369	2018.11

Table 2: statistics of OAG paper data

Data set	#Pairs/Authors	Date
Linking relations	1,717,680	2019.01
AMiner authors	113,171,945	2018.07
MAG authors	253,144,301	2018.11

Open Academic Graph

Open Academic Graph (OAG) is a large knowledge graph unifying two billion-scale academic graphs: Microsoft Academic Graph (MAG) and AMiner. In mid 2017, we published OAG v1, which contains 166,192,182 papers from MAG and 154,771,162 papers from AMiner (see below) and generated 64,639,608 linking (matching) relations between the two graphs. This time, in OAG v2, author, venue and newer publication data and the corresponding matchings are available.

Overview of OAG v2

The statistics of OAG v2 is listed as the three tables below. The two large graphs are both evolving and we take MAG November 2018 snapshot and AMiner July 2018 or January 2019 snapshot for this version.

Heterogeneous Graph Benchmark (HGB)

- A unified benchmark datasets and evaluation pipelines for heterogeneous graph research.
- **Paper:** Are we really making much progress? Revisiting, benchmarking and refining heterogeneous graph neural networks. *(KDD'21)*
- **Code & Data:** https://github.com/THUDM/HGB
- **Leaderboard:** https://www.biendata.xyz/hgb/
- There is also a simple baseline Simple-HGN in HGB. We find that a rather simple design of heterogeneous GNN can reach SOTA.

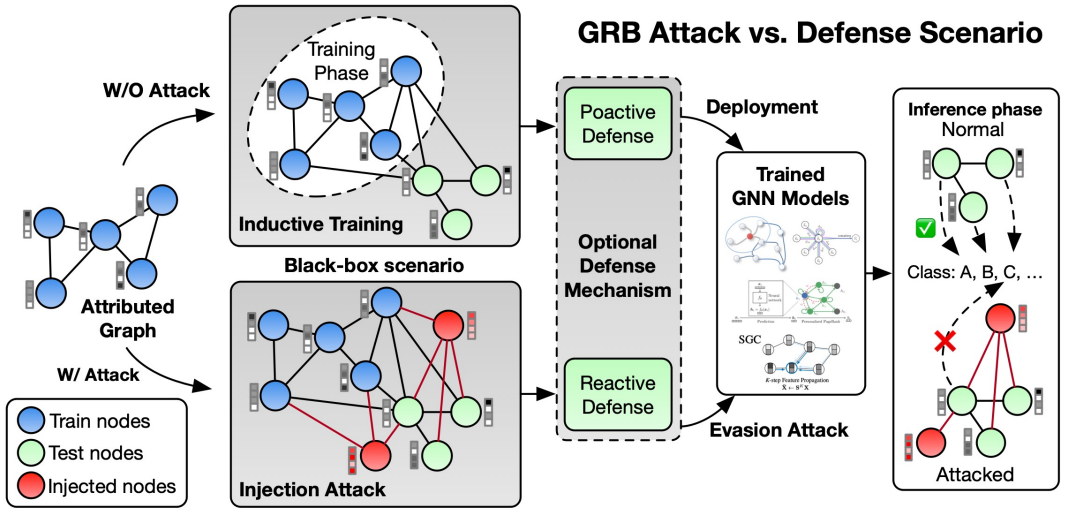
GRB
GRAPH ROBUSTNESS BENCHMARK

Background:

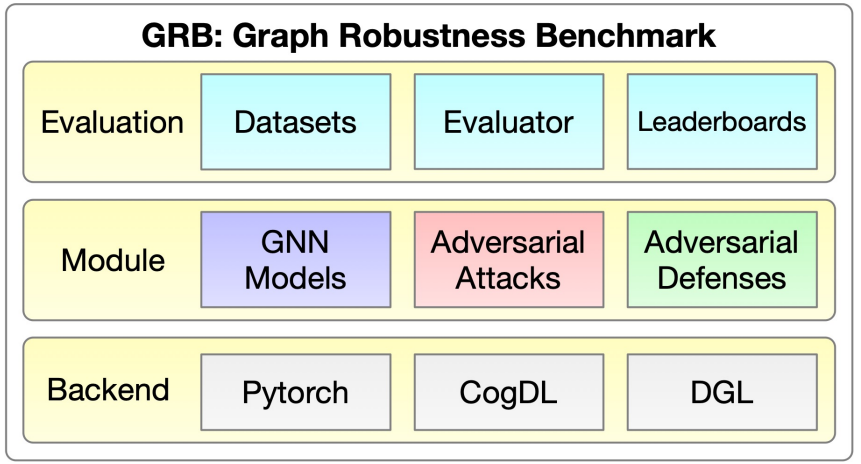
Recently, works have proved that adversarial attacks can threat the *robustness* of graph ML models in various tasks.

Problems:

1. Ill-defined threat model in previous works.
2. Absence of unified and standard evaluation approach.



Example of GRB evaluation scenario



GRB framework

Solution: **Graph Robustness Benchmark (GRB)**

Scalable, *general*, *unified*, and *reproducible* benchmark on *adversarial robustness* of graph ML models, which facilitates fair comparisons among various attacks & defenses and promotes future research in this field.

Graph Robustness Benchmark: Rethinking and Benchmarking Adversarial Robustness of Graph Neural Networks
Qinkai Zheng, Xu Zou, Yuxiao Dong, Yukuo Cen, Jie Tang

GRB
GRAPH ROBUSTNESS BENCHMARK

Graph Robustness Benchmark (GRB): Key Features				
Elaborated Datasets	***Scalability***	Datasets from small to large scales.		
	Specificity	Novel splitting scheme + Preprocessing.		
Modular Framework	***Implementation***	GNNs (GCN, SAGE, GAT, GIN, APPNP, ...)		
		Attacks (FGSM, PGD, SPEIT, TDGIA, ...)		
		Defenses (Adversarial Training, GNNGuard, ...)		
	Backend	Pytorch	CogDL	DGL
Unified Evaluation	***Scenario***	Well-defined realistic threat model.		
	Fairness	Unified settings for attackers and defenders.		
	Pipeline	Easy-to-use evaluation pipeline.		
Reproducible Leaderboard	***Reproducibility***	Availability of methods and related materials.		
	Up-to-date	Continuously maintained to track progress.		

All discussions and contributions are highly welcome!

Homepage:
https://cogdl.ai/grb/home

Github:
https://github.com/THUDM/grb

Leaderboard:
https://cogdl.ai/grb/leaderboard/

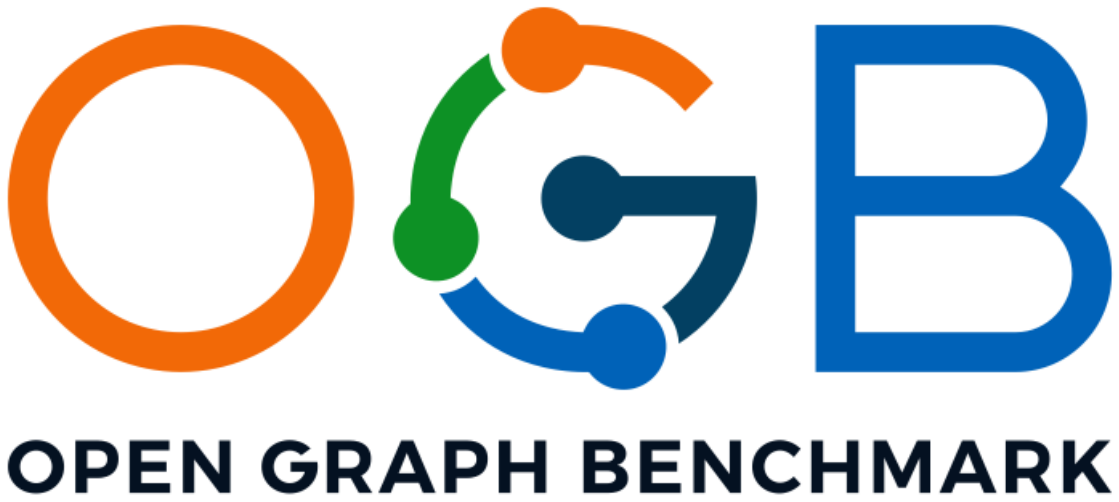
Docs: https://grb.readthedocs.io/

Google Group:
https://groups.google.com/g/graph-robustness-benchmark

Contact:
cogdl.grbteam@gmail.com
qinkai.zheng1028@gmail.com

Open Graph Benchmark

- Large-scale, realistic, and diverse benchmark datasets for graph ML.



Paper: https://arxiv.org/abs/2005.00687
Webpage: https://ogb.stanford.edu/
Github: https://github.com/snap-stanford/ogb

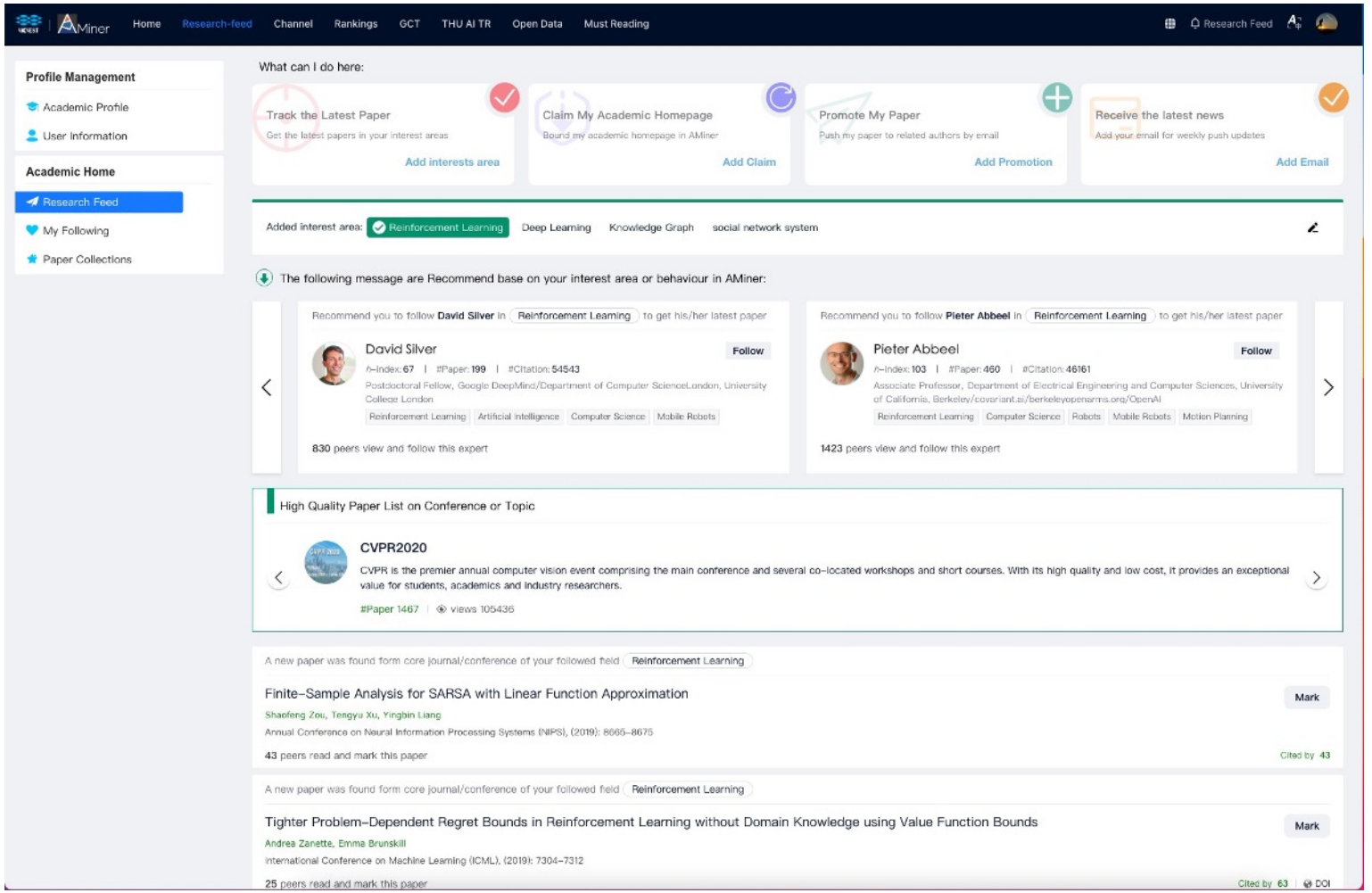
Recommendation Application

- Build recommendation via pipeline API
- Integrate LightGCN (SIGIR'20)
- Similar to Amazon Personalize

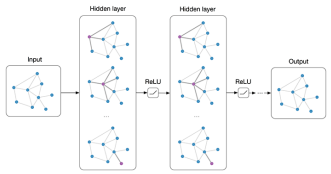
```
import numpy as np
from cogdl import pipeline

data = np.array([[0, 0], [0, 1], [0, 2], [1, 1], [1, 3], [1, 4], [2, 4], [2, 5], [2, 6]])
rec = pipeline("recommendation", model="lightgcn", data=data, max_epoch=1)
print(rec([0]))

rec = pipeline("recommendation", model="lightgcn", dataset="ali", max_epoch=1)
print(rec([0]))
```

AMiner Subscribe

- Recommend papers, scholars to users

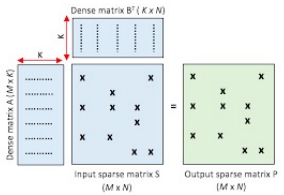


CogDL & dgSPARSE

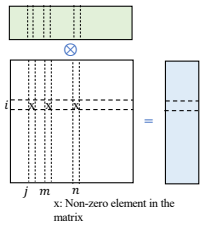


GNNs

- Experiment API
- Validate ideas



- Operator APIs
- Fast deployment



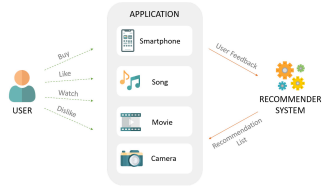
Ops

CogDL

- Fast computation
- Fast development

dgSPARSE

Apps



- Pipeline API
- Deploy Apps

- Sparse APIs
- Hardware support

Hardware

GRL: NE&GNN

Network Embedding ⇔ Matrix Factorization

Graph Neural Networks ⇨ GNN Pre-Training

Learning GNNs with **CogDL**

OPEN GRAPH BENCHMARK

https://alchemy.tencent.com/

Microsoft Academic

Miner

CogDL

https://github.com/thudm/cogdl

Related Publications

For more, check http://keg.cs.tsinghua.edu.cn/jietang

- Jialin Zhao, Yuxiao Dong, Ming Ding, Evgeny Kharlamov, and Jie Tang. Adaptive Diffusion in Graph Neural Networks. **NeurIPS'21**.
- Wenzheng Feng, Jie Zhang, Yuxiao Dong, Yu Han, Huanbo Luan, Qian Xu, Qiang Yang, Evgeny Kharlamov, and Jie Tang. Graph Random Neural Networks for Semi-Supervised Learning on Graphs. **NeurIPS'20**.
- Ming Ding, Chang Zhou, Hongxia Yang, and Jie Tang. CogLTX: Applying BERT to Long Texts. **NeurIPS'20**.
- Jiezhong Qiu, Chi Wang, Ben Liao, Richard Peng, and Jie Tang. Concentration Bounds for Co-occurrence Matrices of Markov Chains. **NeurIPS'20**.
- Xu Zou, Qinkai Zheng, Yuxiao Dong, Xinyu Guan, Evgeny Kharlamov, Jialiang Lu, and Jie Tang. TDGIA: Effective Injection Attacks on Graph Neural Networks**. KDD'21**.
- Qingsong Lv, Ming Ding, Qiang Liu, Yuxiang Chen, Wenzheng Feng, Siming He, Chang Zhou, Jian-guo Jiang, Yuxiao Dong, and Jie Tang. Are we really making much progress? Revisiting, benchmarking and refining the Heterogeneous Graph Neural Networks. **KDD'21**.
- Tinglin Huang, Yuxiao Dong, Ming Ding, Zhen Yang, Wenzheng Feng, Xinyu Wang, and Jie Tang. MixGCF: An Improved Training Method for Graph Neural Network-based Recommender Systems. **KDD'21**.
- Jiezhong Qiu, Qibin Chen, Yuxiao Dong, Jing Zhang, Hongxia Yang, Ming Ding, Kuansan Wang, and Jie Tang. GCC: Graph Contrastive Coding for Structural Graph Representation Pre-Training. **KDD'20**.
- Zhen Yang, Ming Ding, Chang Zhou, Hongxia Yang, Jingren Zhou, and Jie Tang. Understanding Negative Sampling in Graph Representation Learning. **KDD'20**.
- Yukuo Cen, Jianwei Zhang, Xu Zou, Chang Zhou, Hongxia Yang, and Jie Tang. Controllable Multi-Interest Framework for Recommendation. **KDD'20**.
- Yukuo Cen, Xu Zou, Jianwei Zhang, Hongxia Yang, Jingren Zhou and Jie Tang. Representation Learning for Attributed Multiplex Heterogeneous Network. **KDD'19**.
- Fanjin Zhang, Xiao Liu, Jie Tang, Yuxiao Dong, Peiran Yao, Jie Zhang, Xiaotao Gu, Yan Wang, Bin Shao, Rui Li, and Kuansan Wang. OAG: Toward Linking Large-scale Heterogeneous Entity Graphs. **KDD'19**.
- Qibin Chen, Junyang Lin, Yichang Zhang, Hongxia Yang, Jingren Zhou and Jie Tang. Towards Knowledge-Based Personalized Product Description Generation in E-commerce. **KDD'19**.
- Jiezhong Qiu, Jian Tang, Hao Ma, Yuxiao Dong, Kuansan Wang, and Jie Tang. DeepInf: Modeling Influence Locality in Large Social Networks. **KDD'18**.
- Jiezhong Qiu, Laxman Dhulipala, Jie Tang, Richard Peng, and Chi Wang. LightNE: A Lightweight Graph Processing System for Network Embedding. **SIGMOD'21**.
- Yuxiao Dong, Ziniu Hu, Kuansan Wang, Yizhou Sun and Jie Tang. Heterogeneous Network Representation Learning. **IJCAI'20**.
- Jie Zhang, Yuxiao Dong, Yan Wang, Jie Tang, and Ming Ding. ProNE: Fast and Scalable Network Representation Learning. **IJCAI'19**.
- Yifeng Zhao, Xiangwei Wang, Hongxia Yang, Le Song, and Jie Tang. Large Scale Evolving Graphs with Burst Detection. **IJCAI'19**.
- Yu Han, Jie Tang, and Qian Chen. Network Embedding under Partial Monitoring for Evolving Networks. **IJCAI'19**.
- Yifeng Zhao, Xiangwei Wang, Hongxia Yang, Le Song, and Jie Tang. Large Scale Evolving Graphs with Burst Detection. **IJCAI'19**.
- Ming Ding, Chang Zhou, Qibin Chen, Hongxia Yang, and Jie Tang. Cognitive Graph for Multi-Hop Reading Comprehension at Scale. **ACL'19**.
- Jiezhong Qiu, Yuxiao Dong, Hao Ma, Jian Li, Chi Wang, Kuansan Wang, and Jie Tang. NetSMF: Large-Scale Network Embedding as Sparse Matrix Factorization. **WWW'19**.
- Jiezhong Qiu, Yuxiao Dong, Hao Ma, Jian Li, Kuansan Wang, and Jie Tang. Network Embedding as Matrix Factorization: Unifying DeepWalk, LINE, PTE, and node2vec. **WSDM'18**.
- Zhenyu Hou, Yukuo Cen, Yuxiao Dong, Jie Zhang, and Jie Tang. Automated Unsupervised Graph Representation Learning. **TKDE'21**.
- Xiao Liu, Fanjin Zhang, Zhenyu Hou, Li Mian, Zhaoyu Wang, Jing Zhang, and Jie Tang. Self-supervised Learning: Generative or Contrastive. **TKDE'21**.
- Shu Zhao, Ziwei Du, Jie Chen, Yanping Zhang, Jie Tang, and Philip S. Yu. Hierarchical Representation Learning for Attributed Networks. **TKDE'21**.

Thank you !

Collaborators:

Jie Zhang, Ming Ding, Jiezhong Qiu, Qibin Chen, Yifeng Zhao, Yukuo Cen, Yu Han, Fanjin Zhang, Xu Zou, Yan Wang, et al. (**THU**)

Yuxiao Dong, Kuansan Wang (**Microsoft**)

Hongxiao Yang, Chang Zhou, Le Song, Jingren Zhou, et al. (**Alibaba**)

Jie Tang, KEG, Tsinghua U

Download all data & Codes

http://keg.cs.tsinghua.edu.cn/jietang
https://keg.cs.tsinghua.edu.cn/cogdl/
https://github.com/THUDM