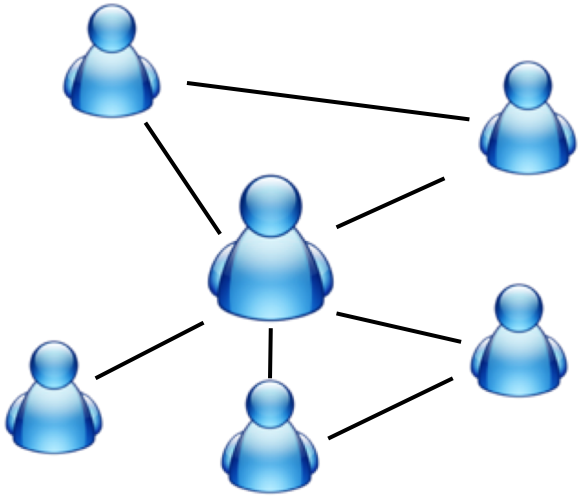
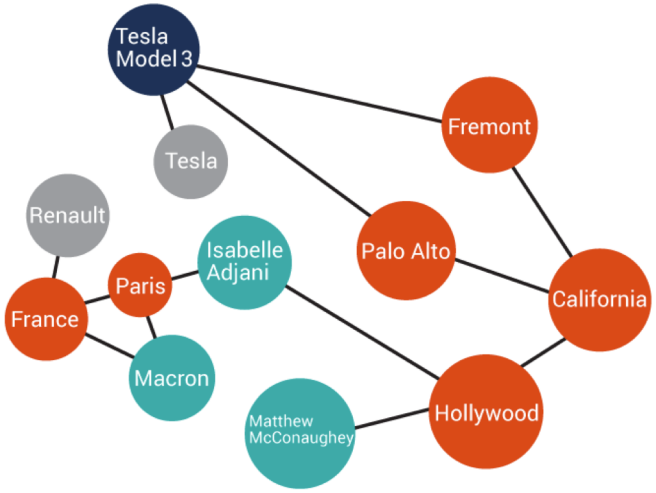
Beyond GNN

Jie Tang
Computer Science
Tsinghua University

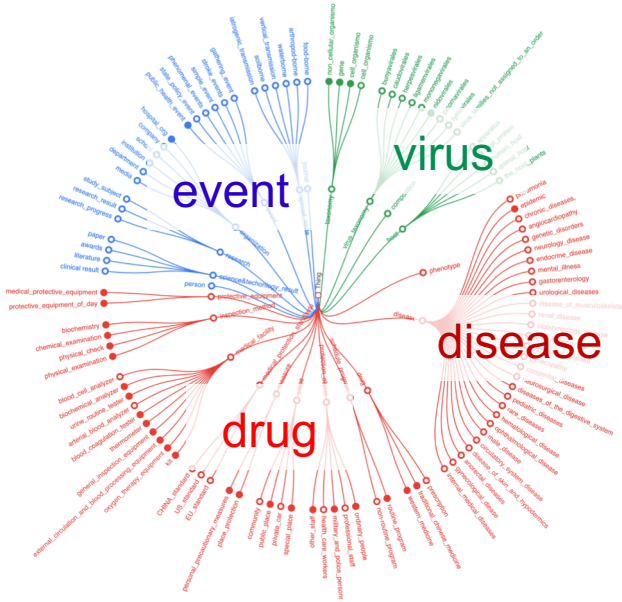
Networked World



Social Network



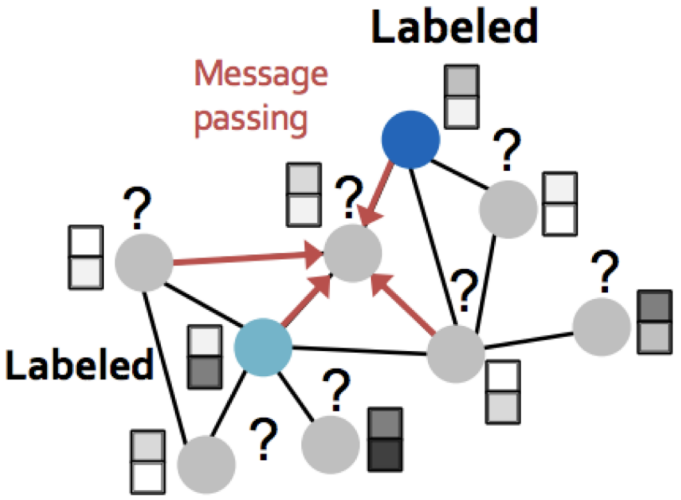
Knowledge Graph



COVID Graph

Setting: Semi-supervised Learning on Graphs

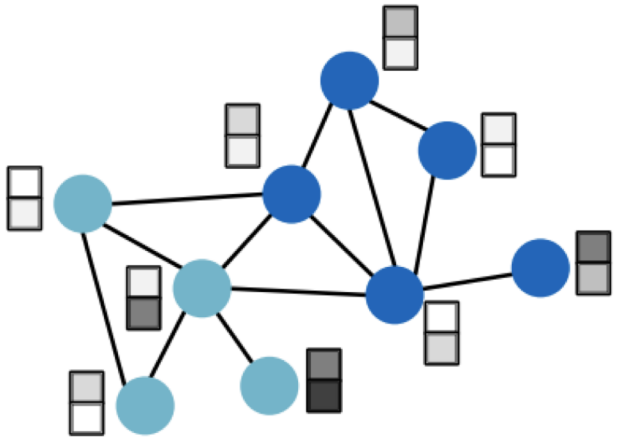
Input: Partially labeled attributed graph



?: Unlabeled

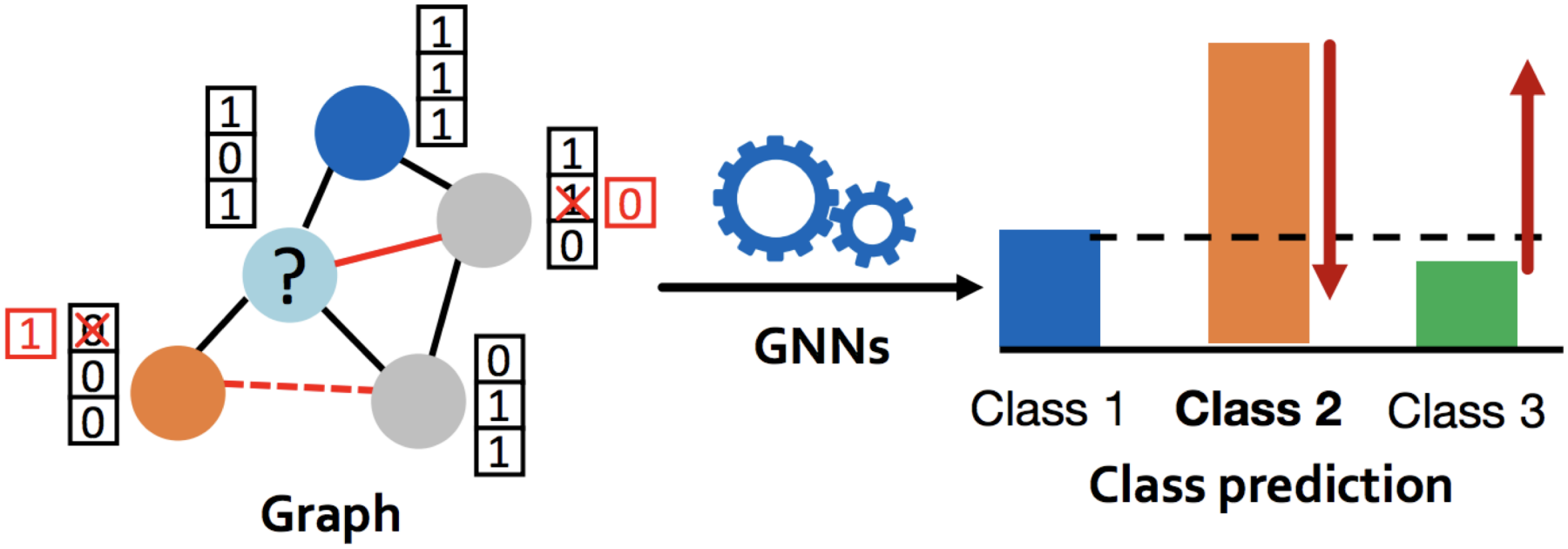
GCN

Goal: Predict labels of unlabeled nodes



Challenges: non-robust

- $\mathbf{H}^{(l+1)} = \sigma(\hat{\mathbf{A}}\mathbf{H}^{(l)}\mathbf{W}^{(l)})$: Deterministic propagation
- Nodes are highly dependent with its neighborhoods making GNNs non-robust.

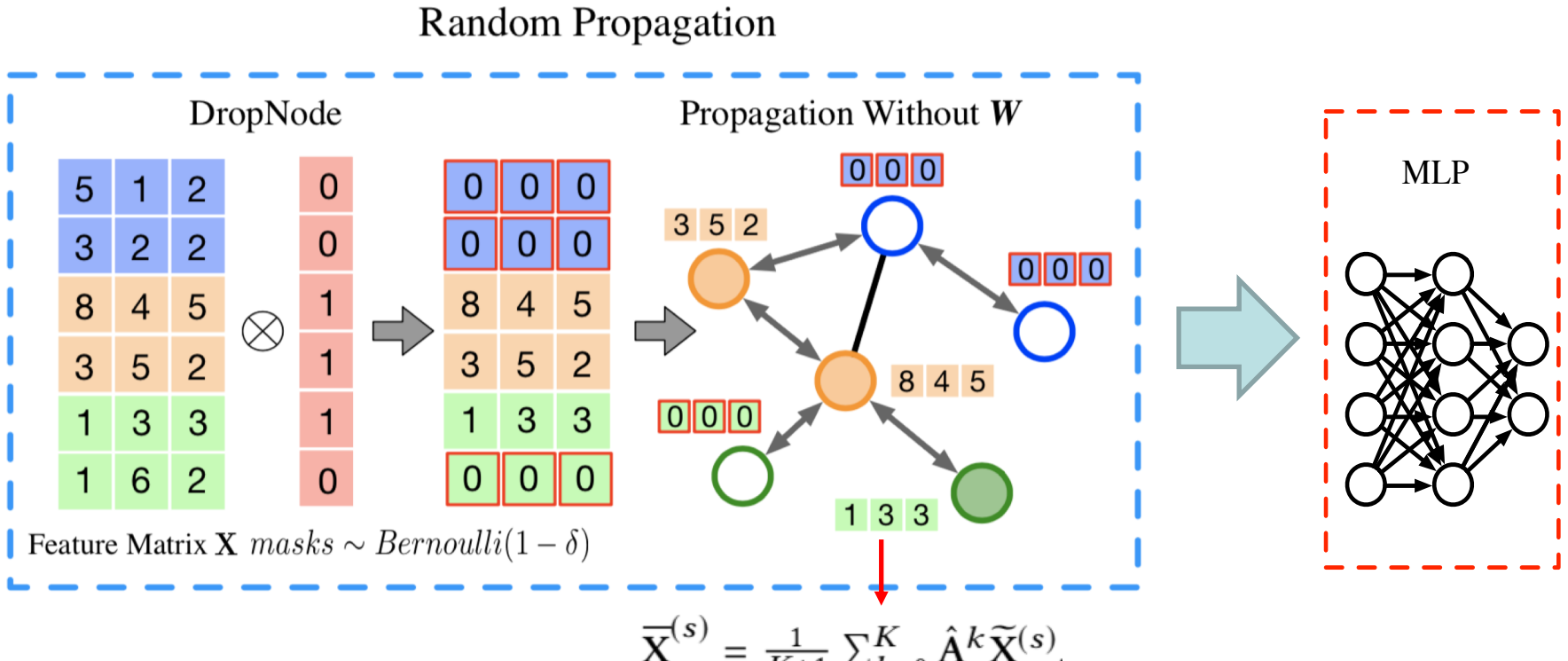


Motivation: some problems of GCN

- $\mathbf{H}^{(l+1)} = \sigma(\hat{\mathbf{A}}\mathbf{H}^{(l)}\mathbf{W}^{(l)})$: Propagation is coupled with transformation.
- Stacking many layers may cause over-fitting and over-smoothing.

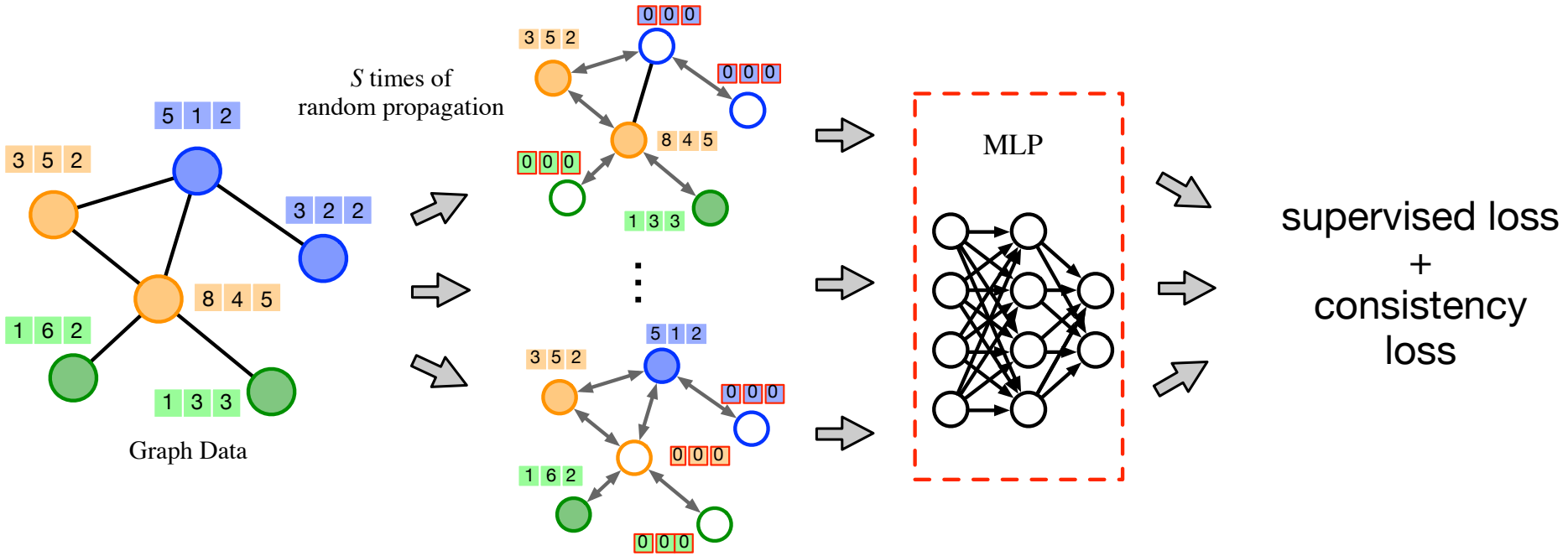
Grand: Graph Random Neural Network

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



Graph Random Neural Network(Grand)

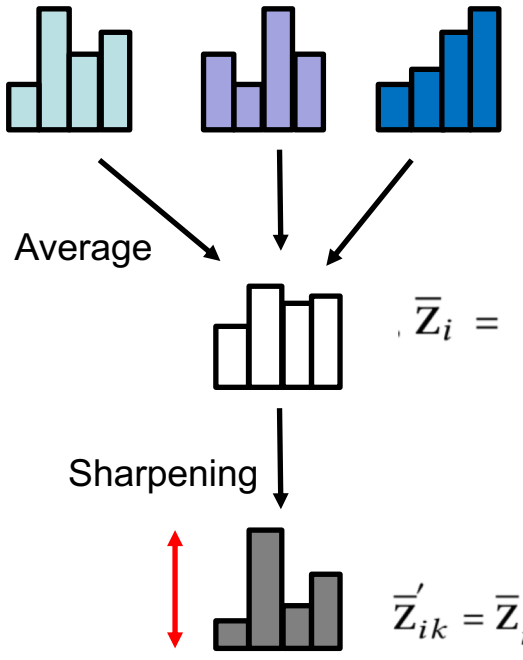
- Consistency Regularized Training:
  - Random propagation serves as graph data augmentation
  - Consistency Regularization: Optimizing the consistency among *S* augmentations of the graph.



Grand training details

Consistency Loss：

Distributions of a node:



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

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

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

Algorithm 2 Consistency Regularized Training for GRAND

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.

Theoretical Analysis

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

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

Different between DropNode and Dropout

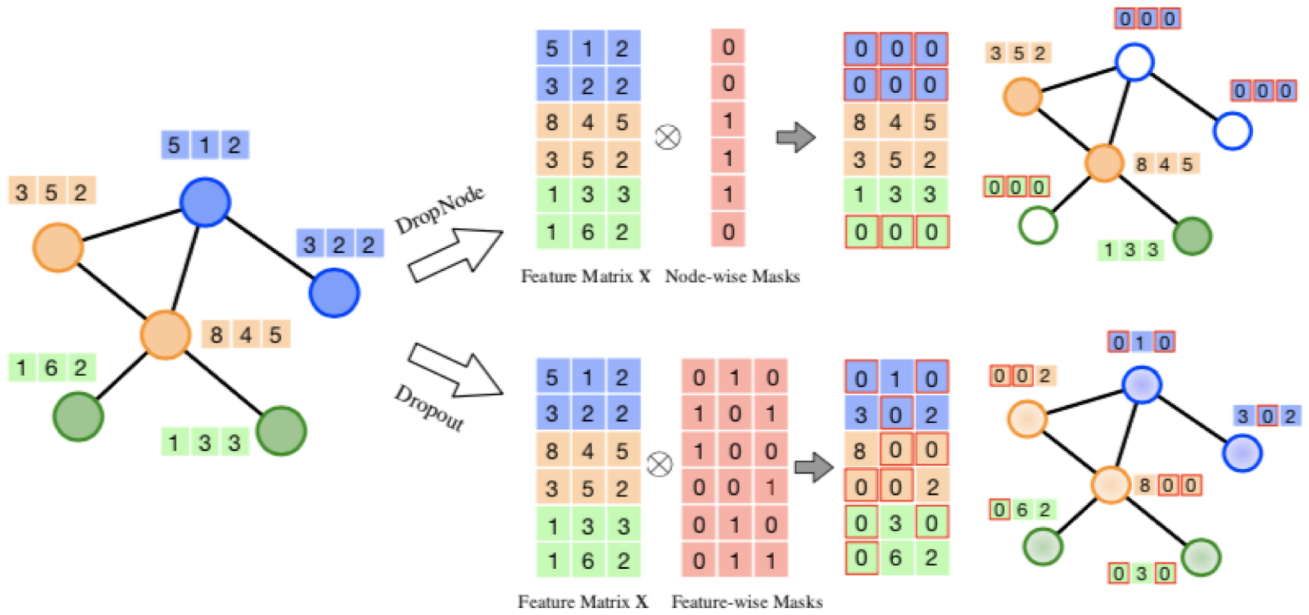


Figure 3: Difference between dropnode and dropout. Dropout drops each element in **X** independently, while DropNode drops the entire features of selected nodes, i.e., the row vectors of **X**, randomly.

- Theoretically, Dropout is an adaptive L2 regularization.

Results

Table 2: Summary of classification accuracy (%).

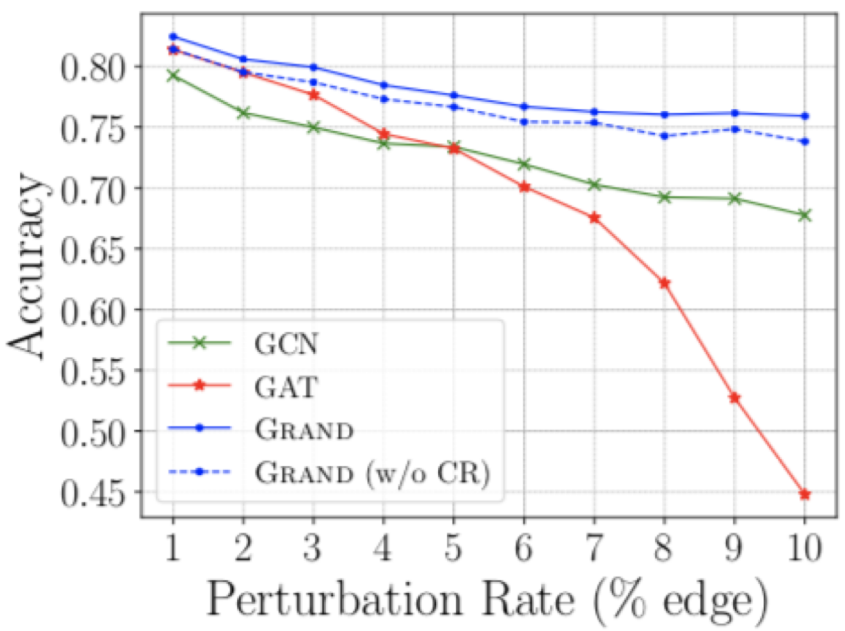
Category	Method	Cora	Citeseer	Pubmed
Graph Convolution	GCN [27]	81.5	70.3	79.0
	GAT [42]	83.0±0.7	72.5±0.7	79.0 ±0.3
	Graph U-Net [15]	84.4±0.6	73.2±0.5	79.6±0.2
	MixHop [2]	81.9 ± 0.4	71.4±0.8	80.8±0.6
	GMNN [36]	83.7	72.9	81.8
	GraphNAS [16]	84.2±1.0	73.1±0.9	79.6±0.4
Regularization based GCNs[2]	VBAT [13]	83.6 ± 0.5	74.0 ± 0.6	79.9 ± 0.4
	G^3NN [31]	82.5 ±0.2	74.4±0.3	77.9 ± 0.4
	GraphMix [43]	83.9 ± 0.6	74.5 ±0.6	81.0 ± 0.6
	DropEdge [37]	82.8	72.3	79.6
Sampling based GCNs[3]	GraphSAGE [22]	78.9±0.8	67.4±0.7	77.8±0.6
	FastGCN [9]	81.4±0.5	68.8±0.9	77.6±0.5
Our methods	GRAND	**85.4±0.4**	**75.4±0.4**	**82.7±0.6**
	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_dropout	84.9±0.4	75.0±0.3	81.7±1.0

Ablation Study

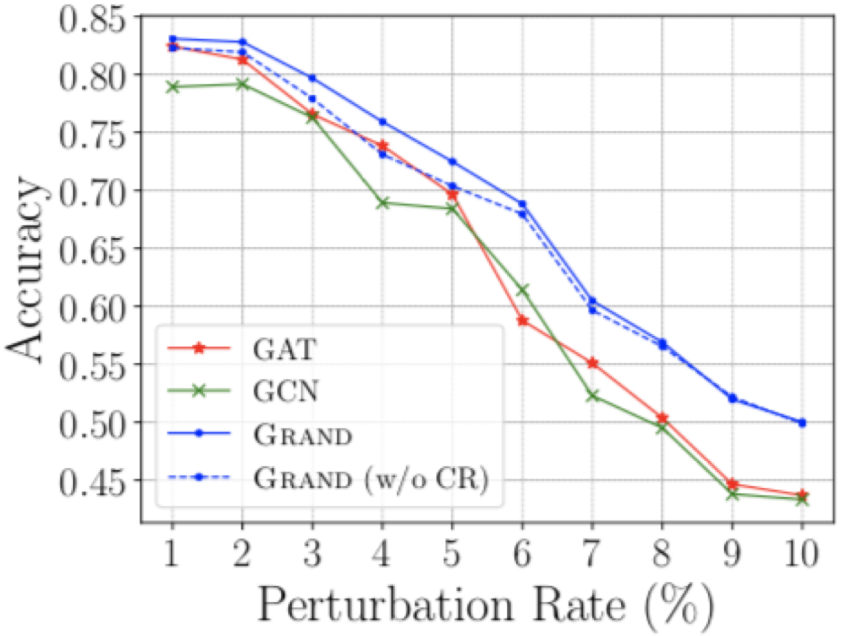
Table 3: Ablation study results (%).

Model	Cora	Citeseer	Pubmed
GRAND	**85.4±0.4**	**75.4±0.4**	**82.7±0.6**
without CR	84.4 ±0.5	73.1 ±0.6	80.9 ±0.8
without multiple DropNode	84.7 ±0.4	74.8±0.4	81.0±1.1
without sharpening	84.6 ±0.4	72.2±0.6	81.6 ± 0.8
without CR and DropNode	83.2 ± 0.5	70.3 ± 0.6	78.5± 1.4

Robustness Analysis



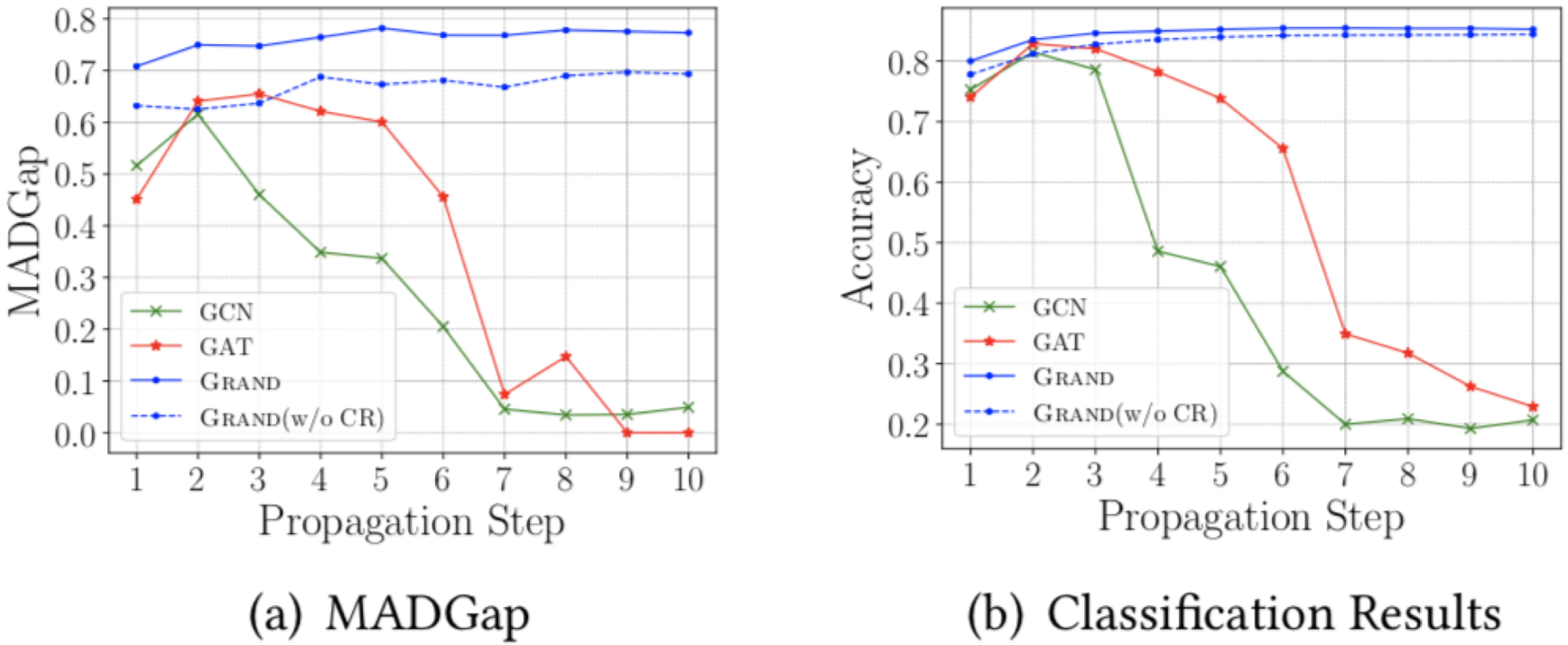
(a) Random Attack



(b) Metattack

Figure 4: Robustness: results under attacks on Cora.

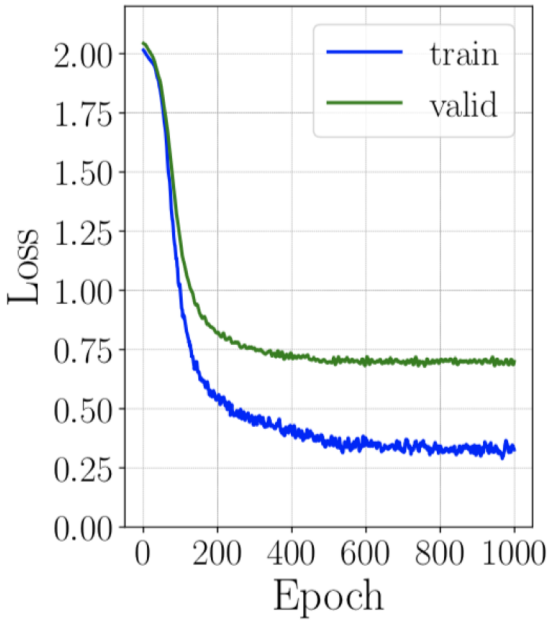
Relieving Over-smoothing



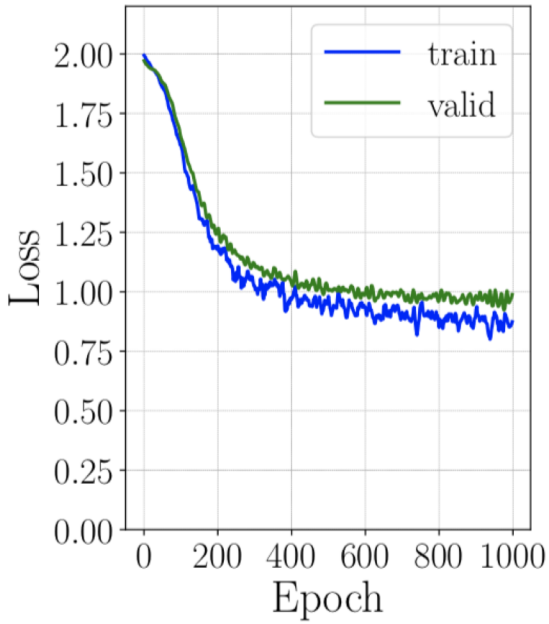
(a) MADGap (b) Classification Results

Figure 5: Over-smoothing: GRAND vs. GCN & GAT on Cora.

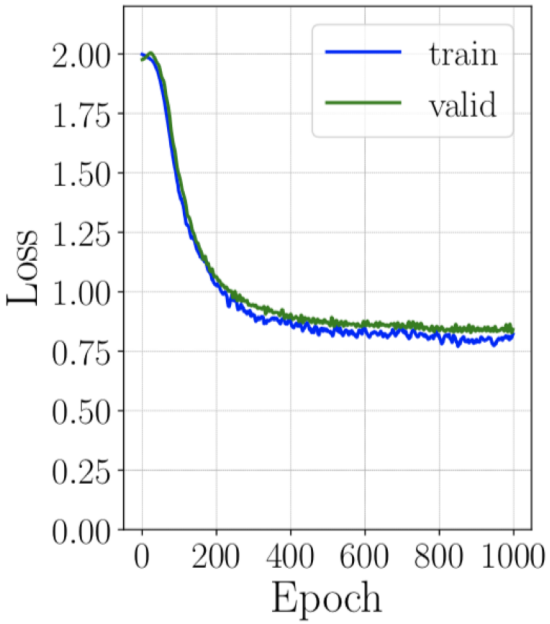
Generalization Improvement



(a) Without CR and RP



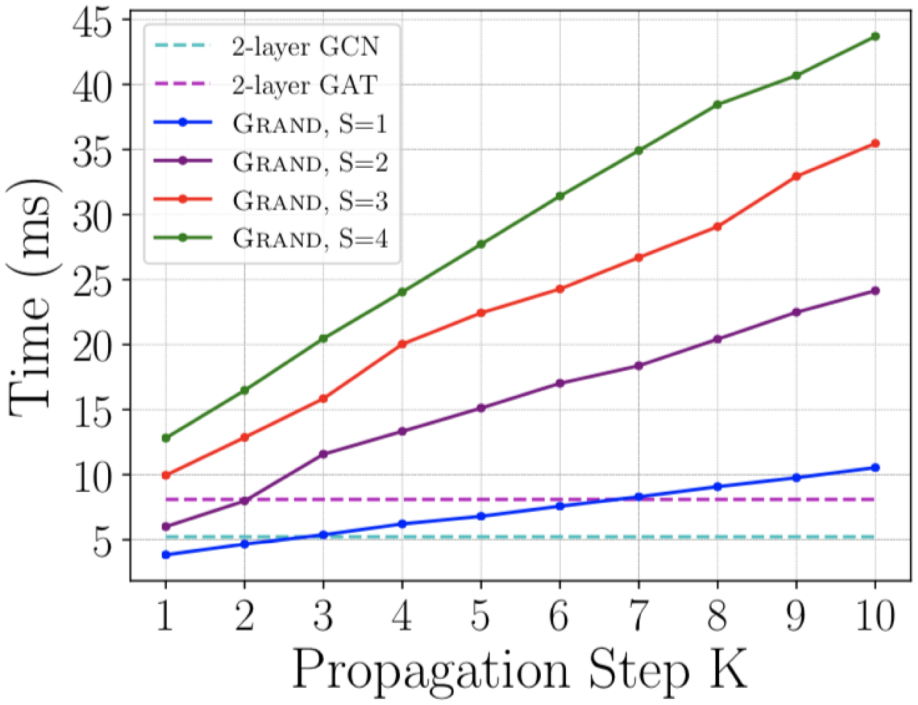
(b) Without CR



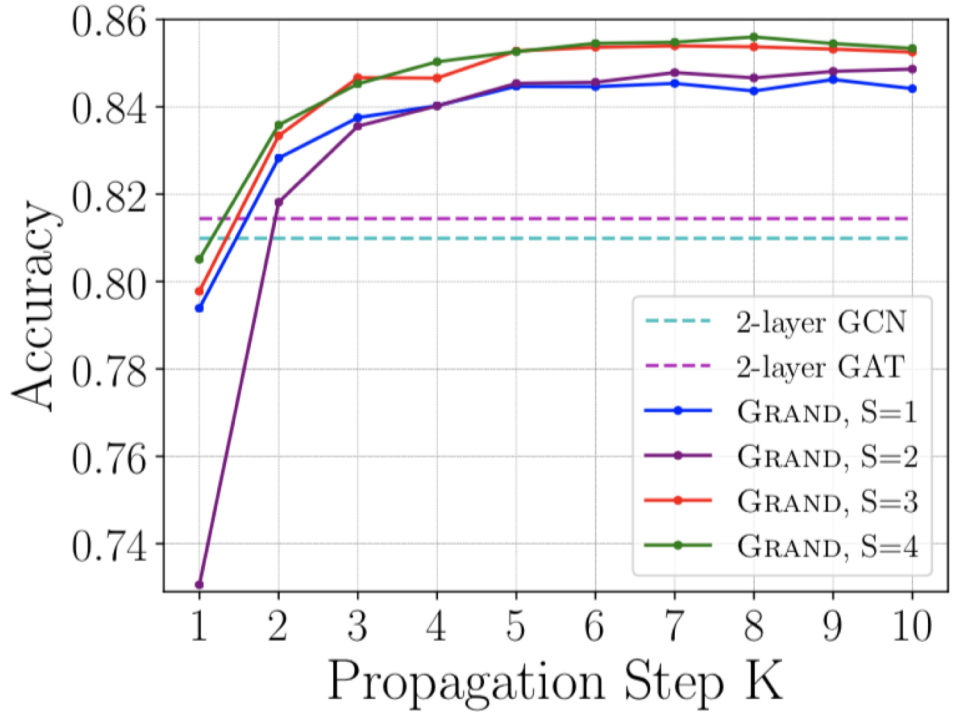
(c) GRAND

Figure 6: Generalization: Training/validation losses on Cora.

Efficiency Analysis



(a) Per-epoch Training Time



(b) Classification Accuracy

Figure 7: Efficiency Analysis for GRAND.

Results on Large Datasets

Table 6: Results on large datasets.

Method	Cora Full	Coauthor CS	Coauthor Physics	Amazon Computer	Amazon Photo	AMiner CS
GCN	62.2 ± 0.6	91.1 ± 0.5	92.8 ± 1.0	82.6 ± 2.4	91.2 ± 1.2	49.4 ± 1.7
GAT	51.9 ± 1.5	90.5 ± 0.6	92.5 ± 0.9	78.0 ± 19.0	85.7 ± 20.3	49.7 ± 1.6
GRAND	**63.5 ±0.6**	**92.9 ± 0.5**	**94.6 ± 0.5**	**85.7 ± 1.8**	**92.5 ± 1.7**	**52.2 ± 1.3**

- Again, what are the fundamentals underlying the different models?

- Understanding GNNs as Signal Rescaling

Case Study - GAT

- Message-Passing as attention coefficients

$$\alpha_{ij} = \frac{\exp\left(\text{LeakyReLU}\left(\vec{a}^T[\mathbf{W}\vec{h}_i||\mathbf{W}\vec{h}_j]\right)\right)}{\sum_{k,\widetilde{A}_{ik}\neq 0}\exp\left(\text{LeakyReLU}\left(\vec{a}^T[\mathbf{W}\vec{h}_i||\mathbf{W}\vec{h}_k]\right)\right)}$$

split $\vec{a}$ *into* $\vec{p}$ *and* $\vec{q}$

$$\alpha_{ij} = \frac{\widetilde{A}_{ij}\cdot\exp\left(\text{LeakyReLU}\left((\mathbf{WH}\vec{p})_i + (\mathbf{WH}\vec{q})_j\right)\right)}{\sum_k \widetilde{A}_{ik}\cdot\exp\left(\text{LeakyReLU}\left((\mathbf{WH}\vec{p})_i + (\mathbf{WH}\vec{q})_k\right)\right)}$$

convert to matrix form

$$\begin{cases}\mathbf{H}' = \sigma\left(\mathcal{N}\left(\mathbf{P}\mathcal{L} + \mathcal{L}\mathbf{Q}\right)\mathbf{HW}\right)\\ \mathcal{L} = A + I_N\\ \mathcal{N}(\cdot) = \text{SparseRowSoftmax}(\text{LeakyReLU}(\cdot))\\ \mathbf{P}_{ii} = (\mathbf{HW}\vec{p})_i\,,\ \mathbf{Q}_{ii} = (\mathbf{HW}\vec{q})_i\end{cases}$$

Case Study – GraphSAGE-mean

- Message-Passing as heuristic sampling and aggregating scheme

$$h_v^k \leftarrow \sigma\left(\mathbf{W} \cdot \text{MEAN}\left(\{h_v^{k-1}\} \cup \{h_u^{k-1},\ j \in \text{Sample}(i)\}\right)\right)$$

convert to matrix form

$$\begin{cases} \mathbf{H}' = \sigma\left(\mathcal{N}\left(\mathcal{L} \odot \mathbf{Q}\right)\mathbf{H}\mathbf{W}\right) \\ \mathcal{L} = A + I_N \\ \mathcal{N}(\mathbf{M}) = D^{-1}\mathbf{M},\ D_{ii} = \sum_j \mathbf{M}_{ij} \\ \left\{\mathbf{Q}_{ij} | \hat{A}_{ij} \neq 0\right\} \sim \text{Sample}\left(\deg(i), S_l\right) \end{cases}$$

Case Study - FastGCN

- Message-Passing as heuristic sampling and aggregating scheme

$$\begin{cases} \mathbf{H}^{(l+1)}(v,:) = \sigma\left(\frac{1}{t_l}\sum_{j=1}^{t_l}\frac{\hat{A}\left(v,u_j^{(l)}\right)\mathbf{H}^{(l)}\left(u_j^{(l)},:\right)\mathbf{W}^{(l)}}{q(u_j^{(l)})}\right) \\ q(u) = ||\hat{A}(:,u)||^2 / \sum_{u'\in V}||\hat{A}(:,u')||^2, u \in V \end{cases}$$



convert to matrix form

$$\begin{cases} \mathbf{H}' = \sigma\left(\mathcal{N}\left(\mathcal{L}\mathbf{Q}\right)\mathbf{H}\mathbf{W}\right) \\ \mathcal{N}(\mathbf{M}) = \frac{1}{t_l}\mathbf{M} \\ \{q(u_1)\mathbf{Q}_{11}, q(u_2)\mathbf{Q}_{22}, \cdots, q(u_n)\mathbf{Q}_{nn}\} \sim \text{Sample}(N, t_l) \end{cases}$$

Case Study - ASGCN

- Message-Passing as learned sampling and aggregating scheme

$$q^*(u_j) = \frac{\sum_{i=1}^{n} p(u_j|v_i)|g(x(u_j))|}{\sum_{k=1}^{N}\sum_{i=1}^{n} p(u_k|v_i)|g(x(u_k))|}$$

$$h^{(l+1)}(v_i) = \sigma\left(N(v_i)\frac{1}{n}\sum_{j=1}^{n}\frac{p(u_j|v_i)}{q^*(u_j)}h^{(l)}(u_j)\right)$$

$$u_j \sim q^*(u_j)$$

convert to matrix form

$$\begin{cases} \mathbf{H}' = \sigma\left(\mathcal{N}\left(\mathcal{L}\mathbf{Q}\right)\mathbf{H}\mathbf{W}\right) \\ \mathcal{N}(\mathbf{M}) = \frac{1}{n}\mathbf{M} \\ \mathbf{Q}_{ii} = \frac{q_i}{q^*(u_i)},\ q_i \sim \mathrm{Ber}(n, q^*(u_i)) \end{cases}$$

Summarization

Methods	Rescaling and Propagation Rules
vanilla GCN	$\mathbf{H}' = \sigma\left(\mathcal{L}\mathbf{H}\mathbf{W}\right)$
GAT	$\mathbf{H}' = \sigma\left(\mathcal{N}\left(\mathbf{P}\mathcal{L} + \mathcal{L}\mathbf{Q}\right)\mathbf{H}\mathbf{W}\right)$
$\text{GraphSAGE}_{\text{mean}}$	$\mathbf{H}' = \sigma\left(\mathcal{N}\left(\mathcal{L} \odot \mathbf{Q}\right)\mathbf{H}\mathbf{W}\right)$
FastGCN	$\mathbf{H}' = \sigma\left(\mathcal{N}\left(\mathcal{L}\mathbf{Q}\right)\mathbf{H}\mathbf{W}\right)$
ASGCN	$\mathbf{H}' = \sigma\left(\mathcal{N}\left(\mathcal{L}\mathbf{Q}\right)\mathbf{H}\mathbf{W}\right)$

- P and Q are rescaling coefficients matrices.
- $\mathcal{N}(\cdot)$ denotes some matrix normalization function.
- $\mathcal{L}$ is the model specified Laplacian matrix.
- $\sigma(\cdot)$ can be any activation function.

Observations

- Propagation – by Laplacian
  - Laplacians are normalized adjacency matrix with self-loop
  - $\mathcal{L}$ can be $D^{-1}(A+I), D^{-\frac{1}{2}}(A+I)D^{-\frac{1}{2}}$ or even $A+I$.
- Signal Rescaling
  - Before Propagation (Pre-Prop)
  - After Propagation (Post-Prop)
  - During Propagation (Edge-Wise)
  - Re-normalization (For numerical stability)

A combination gives approximation for

Observations

- In this way, we
  - **disentangle** rescaling from message-passing
  - **decouple** rescaling operations into different operators

- Based on our observation, we are able to design a general / flexible GCN framework with decoupled rescaling operators

Building Blocks of Signal Rescaling

- Pre-propagation Rescaling
  - Vanilla GCN: $H' = \sigma(\mathcal{L}HW)$
  - Matrix row scaling operator as Diagonal matrix P and Q
  - Rescaling before propagation: $H' = \sigma(\mathcal{L}QHW)$
  - Re-normalization after aggregation: $H' = \sigma(P\mathcal{L}QHW)$
  - Q are extracted from pre-prop feature by a learnable vector $\vec{p}$: $Q_{ii} = \text{sigmoid}(HW\vec{q})_i$
  - P makes the row sum of $P\mathcal{L}Q$ to be 1.

Building Blocks of Signal Rescaling

- Post-propagation Rescaling
  - Vanilla GCN: $H' = \sigma(\mathcal{L}HW)$
  - Negative signal rescaling by node adaptive encoder:

$$\sigma_{\vec{r}}(x) = \begin{cases} \sigma(x), x \geq 0 \\ \sigma(\vec{r}_i x), x < 0 \end{cases}$$

  - Rescaling after propagation $H' = \sigma_{\vec{r}}(\mathcal{L}HW)$

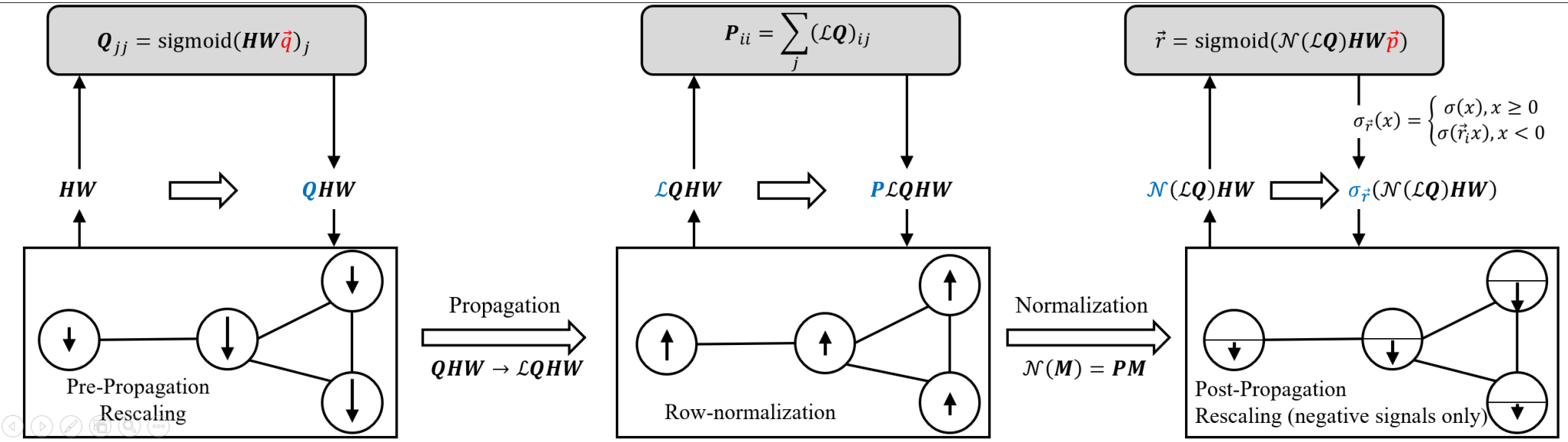
  - $\vec{r}$ are extracted from post-prop feature by a learnable vector $\vec{p}$: $\vec{r}_i = \mathrm{sigmoid}(\mathcal{L}HW\vec{q})_i$
  - Re-normalization are opted out for $\mathrm{xLU}(\cdot)$ activation's output mostly concentrate on the positive signal.

Building Blocks of Signal Rescaling

- Edge-wise Rescaling
  - Approximated by the combination of Pre-prop rescaling and Post-prop rescaling

- Node Sampling Strategy
  - (Schematic) Dropout applied to Pre-propagation Matrix

General Framework



Diagonal Matrix Q as Pre-prop rescaling operator
Node Adaptive Encoder $\sigma_{\vec{r}}(\cdot)$ as Post-prop rescaling operator

Analysis

- The pre-propagation rescaling redistributes the signal energy. It functions as noise reducer.
- The post-propagation rescaling reduce the variance before activation. It functions like Layer Normalization.
- The Node Sampling Strategy functions like small batch trick (Regularization).
- The flexibility of Signal Rescaling framework is favorable by Graph AutoML.

Graph Attention as Signal Rescaling

- GAT
  - $H' = \sigma(\mathcal{N}(P\mathcal{L} + \mathcal{L}Q)HW)$
  - $\mathcal{L} = A + I_N$
  - $\mathcal{N}(\cdot) = \mathrm{SparseRowSoftmax}\big(\mathrm{LeakyReLU}(\cdot)\big)$
  - $P_{ii} = (HW\vec{p})_i\,, \quad Q_{ii} = (HW\vec{q})_i$

Graph Attention as Signal Rescaling

- Node Attention
  - $H_i' = \sigma_i(P\mathcal{L}QHW)_i$
  - $P_{ii} = \frac{1}{\vec{q}\cdot\vec{1}}$
- Edge Attention
  - $H_i' = \sigma_i(P\mathcal{L}QHW)_i$
  - $P_{ii} = \frac{1}{\mathcal{L}Q\vec{1}}$
- k-hop Edge Attention
  - $H_i' = \sigma_i\left(P\mathcal{L}^kQHW\right)_i$
  - $P_{ii} = \frac{1}{\mathcal{L}^kQ\vec{1}}$
- k-hop Path Attention
  - $H_i' = \sigma_i(P\mathcal{L}Q_k\mathcal{L}Q_{k-1}\cdots\mathcal{L}Q_1HW)_i$
  - $P_{ii} = \frac{1}{\mathcal{L}Q_k\mathcal{L}Q_{k-1}\cdots\mathcal{L}Q_1\vec{1}}$

Performance – Citation Networks

Category	Method	Cora	Citeseer	Pubmed
Graph Convolution	Chebyshev	81.2	69.8	74.4
	GCN	81.5	70.3	79.0
	MixHop	81.9±0.4	71.4±0.8	**80.8±0.6**
	FastGCN*	80.0±0.6	69.5±0.6	77.8±0.4
	ASGCN*	78.5±0.5	68.9±0.5	75.5±0.4
	GAT	83.0±0.7	72.5±0.7	79.0±0.3
	GAT-SR**	83.4±0.4	72.4±0.6	78.7±0.4
	GAT-16**	83.4±0.4	72.3±0.6	78.8±0.3
	SRGCN	**84.1±0.5**	**73.4±0.7**	79.5±0.4
	SRGCN (Pre)	83.8±0.6	73.2±0.6	79.3±0.6
	SRGCN (Post)	**84.1±0.5**	73.3±0.7	79.4±0.5
	SRGCN (1/2)	83.9±0.6	73.3±0.9	79.3±0.6
	SRGCN (1/4)	84.0±0.6	73.0±0.9	79.0±0.8
	SRGCN (1/8)	84.0±0.6	72.5±1.4	77.5±1.3

Performance – PPI networks

Method	F1	Method	F1
Random	39.6	GeniePath	97.9
MLP	42.2	GAT	97.3±0.2
GraphSAGE-GCN	50.0	SRGCN	**98.4±0.2**
GraphSAGE-mean	59.8	SRGCN (Pre)	98.4±0.3
GraphSAGE-LSTM	61.2	SRGCN (Post)	97.3±0.2
GraphSAGE-pool	60.0	SRGCN (1/2)	98.1±0.4
GraphSAGE	76.8	SRGCN (1/4)	96.7±0.5

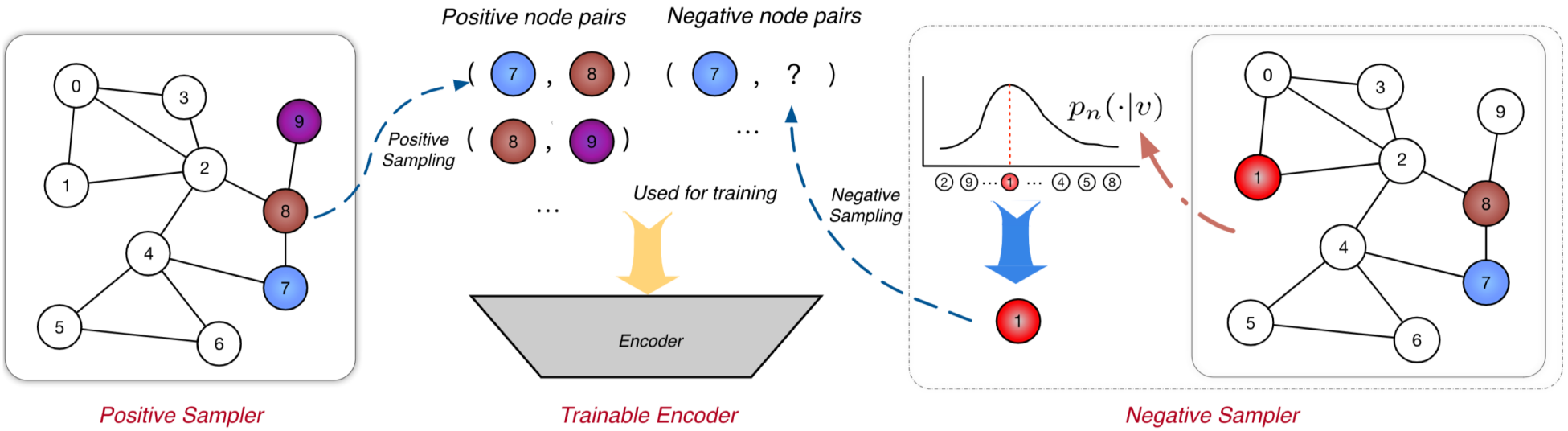
Takeaway Messages & Future Works

- The decoupled rescaling operations have their separated utilities.
- Node sampling strategy functions as small batch trick.
  - Can we find a theory?
  - How to better use this?
- The graph AutoML has its potential to further enhance of the community of GNNs.
  - Realization?

- Let us now revisit the negative sampling?

SampledNCE Framework

- An trainable encoder E_θ-- generate node embeddings
  - An embedding lookup table
  - A variant of Graph Neural Networks
- A positive sampler -- sample positive nodes
- A negative sampler -- sample negative nodes



1. Z. Yang, M. Ding, C. Zhou, H. Yang, J. Zhou, and J. Tang. Understanding Negative Sampling in Graph Representation Learning. *KDD'20*.

Negative Sampling

- Negative Sampling
  - serve as a simplified version of noise contrastive estimation.
  - a efficient way to accelerate the training of word2vec.
  - negative sampling has a huge influence on the performance.
  - the best choice varies largely on different datasets.
  - few papers systematically analyzed or discussed negative sampling in graph representation learning.

Related Work

- Degree-based NS: utilize degree to sample negative samples
  - Power of Degree (Mikolov, Sutskever & Dean, 2013)
  - RNS (Caselles-Dupré, Lesaint & Royo-Letelier, 2018)
  - WRMF (Hu, Koren & Volinsky, 2008)
- Hard-samples NS: mine the hard negative samples by rejection
  - DNS (Zhang, Chen & Yu, 2013)
  - PinSAGE (Rex, Ruining& Jure, 2018)
  - WARP (Weston, Bengio & Usunier, 2011)
- GAN-based NS: utilize GAN to adversarially generate “difficult” negative samples
  - IRGAN (Wang, Yu & Xu, 2017)
  - KBGAN (Cai & Wang, 2018)
  - NMRN (Wang, Yin & Wang, 2018)

Understanding Negative Sampling--Objective

The objective function for embedding:

$$J = \mathbb{E}_{(u,v)\sim p_d} \log \sigma(\vec{\boldsymbol{u}}^T \vec{\boldsymbol{v}}) + \mathbb{E}_{v\sim p_d(v)}[k\mathbb{E}_{u'\sim p_n(u'|v)} \log \sigma(-\vec{\boldsymbol{u}}'^T \vec{\boldsymbol{v}})]$$

For each (u, v) pair, define two Bernoulli distribution:

- P: $P_{u,v}(x = 1) = \frac{p_d(u|v)}{p_d(u|v) + kp_n(u|v)}$ Q: $Q_{u,v}(x = 1) = \sigma(\vec{\boldsymbol{u}}^T \vec{\boldsymbol{v}})$

The objective function can be simplified as follows:

$$J = -\sum_u (p_d(u|v) + kp_n(u|v)) H(P_{u,v}, Q_{u,v}), \text{where } H(p, q) \text{ is the cross entropy.}$$

According to *Gibbs Inequality*, the optimal embedding for each node pair (u, v) :

$$\vec{\boldsymbol{u}}^T \vec{\boldsymbol{v}} = -\log \frac{k \cdot p_n(u|v)}{p_d(u|v)}$$

Understanding Negative Sampling--Risk

The loss function to minimize *empirical risk* as follows:

$$J_T^{(v)} = \frac{1}{T}\sum_{i=1}^{T}\log\sigma(\vec{\boldsymbol{u}}_i^T\vec{\boldsymbol{v}}) + \frac{1}{T}\sum_{i=1}^{kT}\log\sigma(-\vec{\boldsymbol{u}'}_i^T\vec{\boldsymbol{v}}),$$

$$where\ \{u_1,...,u_T\} are\ sampled\ from\ p_d(u|v)\ and\ \{u'_1,...,u'^{\top}_k\}\ are\ sampled\ from\ p_n(u|v).$$

Theorem: the random variable $\sqrt{T}(\theta_T - \theta^*)$ asymptotically converges to a distribution with zero mean vector and covariance matrix.

$$\mathrm{Cov}\big(\sqrt{T}(\theta_T-\theta^*)\big) = \mathbf{diag}(m)^{-1} - (1+1/k)1^{\top}1$$

$$\text{where } m = \Big[\tfrac{kp_d(u_0|v)p_n(u_0|v)}{p_d(u_0|v)+kp_n(u_0|v)},...,\tfrac{kp_d(u_{N-1}|v)p_n(u_{N-1}|v)}{p_d(u_{N-1}|v)+kp_n(u_{N-1}|v)}\Big]^T \text{ and } 1=[1,...,1]^T.$$

According to Theorem, the mean squared error of $\vec{\boldsymbol{u}}^T\vec{\boldsymbol{v}}$:

$$\mathbb{E}\big[\|(\theta_T-\theta^*)_u\|^2\big] = \frac{1}{T}\Big(\frac{1}{p_d(u|v)} - 1 + \frac{1}{kp_n(u|v)} - \frac{1}{k}\Big)$$

The Principle of Negative Sampling

A simple solution is to sample negative nodes *positively but sub-linearly* correlated to their positive sampling distribution.

$$p_n(u|v) \propto p_d(u|v)^\alpha, 0 < \alpha < 1$$

- Monotonicity: from the perspective of objective, if we have $p_d(u_i|v) > p_d(u_j|v)$,

$$\begin{aligned}\vec{\boldsymbol{u}}_i^T \vec{\boldsymbol{v}} &= \log p_d(u_i|v) - \alpha \log p_d(u_i|v) + c \\ &> (1-\alpha) \log p_d(u_j|v) + c = \vec{\boldsymbol{u}}_j^T \vec{\boldsymbol{v}}\end{aligned}$$

- Accuracy: from the perspective of risk, if $p_n(u|v) \propto p_d(u|v)^\alpha$,

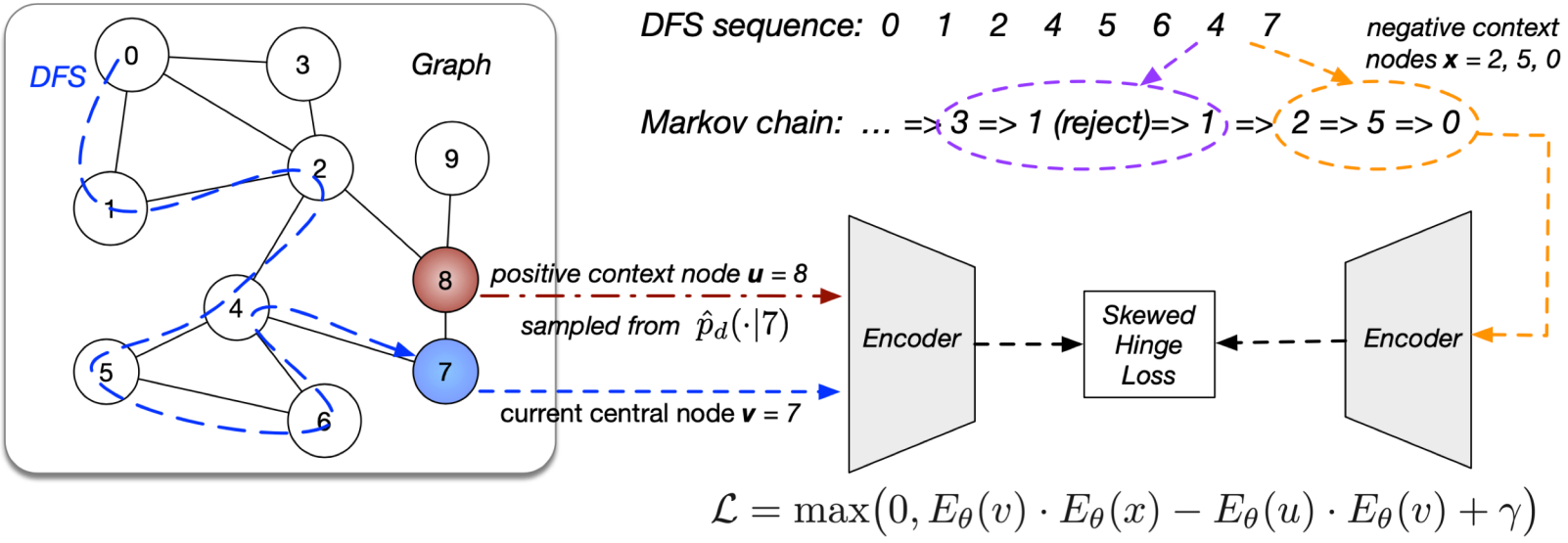
$$\mathbb{E}\left[\|(\theta_T - \theta^*)_u\|^2\right] = \frac{1}{T}\left(\frac{1}{p_d(u|v)}\left(1 + \frac{p_d(u|v)^{1-\alpha}}{c}\right) - 1 - \frac{1}{k}\right)$$

MCNS Model

- self-contrast approximation for positive sampling distribution:
  - replacing p_d by inner products based on the current encoder

$$p_n(u|v) \propto p_d(u|v)^\alpha \approx \frac{\left(E_\theta(u) \cdot E_\theta(v)\right)^\alpha}{\sum_{u' \in U} \left(E_\theta(u') \cdot E_\theta(v)\right)^\alpha}$$

- Metropolis-Hastings to accelerate negative sampling



Experiments

Recommendation Task

MovieLens:

2,625 nodes

100,000 links

Amazon:

255,404 nodes

1,689,188 links

Alibaba:

159,633 nodes

907,470 links

Evaluation Metric:

MRR & Hits@30

		MovieLens			Amazon		Alibaba	
		DeepWalk	GCN	GraphSAGE	DeepWalk	GraphSAGE	DeepWalk	GraphSAGE
MRR	$Deg^{0.75}$	0.025±.001	0.062±.001	0.063±.001	0.041±.001	0.057±.001	0.037±.001	0.064±.001
	WRMF	0.022±.001	0.038±.001	0.040±.001	0.034±.001	0.043±.001	0.036±.001	0.057±.002
	RNS	0.031±.001	0.082±.002	0.079±.001	0.046±.003	0.079±.003	0.035±.001	0.078±.003
	PinSAGE	0.036±.001	0.091±.002	0.090±.002	0.057±.004	0.080±.001	0.054±.001	0.081±.001
	WARP	0.041±.003	0.114±.003	0.111±.003	0.061±.001	0.098±.002	0.067±.001	0.106±.001
	DNS	0.040±.003	0.113±.003	0.115±.003	0.063±.001	0.101±.003	0.067±.001	0.090±.002
	IRGAN	0.047±.002	0.111±.002	0.101±.002	0.059±.001	0.091±.001	0.061±.001	0.083±.001
	KBGAN	0.049±.001	0.114±.003	0.100 ±.001	0.060±.001	0.089±.001	0.065±.001	0.087±.002
	MCNS	**0.053±.001**	**0.122±.004**	**0.114±.001**	**0.065±.001**	**0.108±.001**	**0.070±.001**	**0.116±.001**
Hits@30	$Deg^{0.75}$	0.115±.002	0.270±.002	0.270±.001	0.161±.003	0.238±.002	0.138±.003	0.249±.004
	WRMF	0.110±.003	0.187±.002	0.181±.002	0.139±.002	0.188±.001	0.121±.003	0.227±.004
	RNS	0.143±.004	0.362±.004	0.356±.001	0.171±.004	0.317±.004	0.132±.004	0.302±.005
	PinSAGE	0.158±.003	0.379±.005	0.383±.005	0.176±.004	0.333±.005	0.146±.003	0.312±.005
	WARP	0.164±.005	0.406±.002	0.404±.005	0.181±.004	0.340±.004	0.178±.004	0.342±.004
	DNS	0.166±.005	0.404±.006	0.410±.006	0.182±.003	0.358±.004	0.186±.005	0.336±.004
	IRGAN	0.207±.002	0.415±.004	0.408±.004	0.183±.004	0.342±.003	0.175±.003	0.320±.002
	KBGAN	0.198±.003	0.420±.003	0.401±.005	0.181±.003	0.347±.003	0.181±.003	0.331±.004
	MCNS	**0.230±.003**	**0.426 ±.005**	**0.413±.003**	**0.207±.003**	**0.386±.004**	**0.201±.003**	**0.387±.002**

Experiments

Link Prediction Task

Arxiv: 5,242 nodes & 28,980 links

Evaluation Metric: AUC

	Algorithm	DeepWalk	GCN	GraphSAGE
AUC	$Deg^{0.75}$	64.6±0.1	79.6±0.4	78.9±0.4
	WRMF	65.3±0.1	80.3±0.4	79.1±0.2
	RNS	62.2±0.2	74.3±0.5	74.7±0.5
	PinSAGE	67.2±0.4	80.4±0.3	80.1±0.4
	WARP	70.5±0.3	81.6±0.3	82.7±0.4
	DNS	70.4±0.3	81.5±0.3	82.6±0.4
	IRGAN	71.1±0.2	82.0±0.4	82.2±0.3
	KBGAN	71.6±0.3	81.7±0.3	82.1±0.3
	MCNS	**73.1±0.4**	**82.6±0.4**	**83.5±0.5**

Node Classification Task

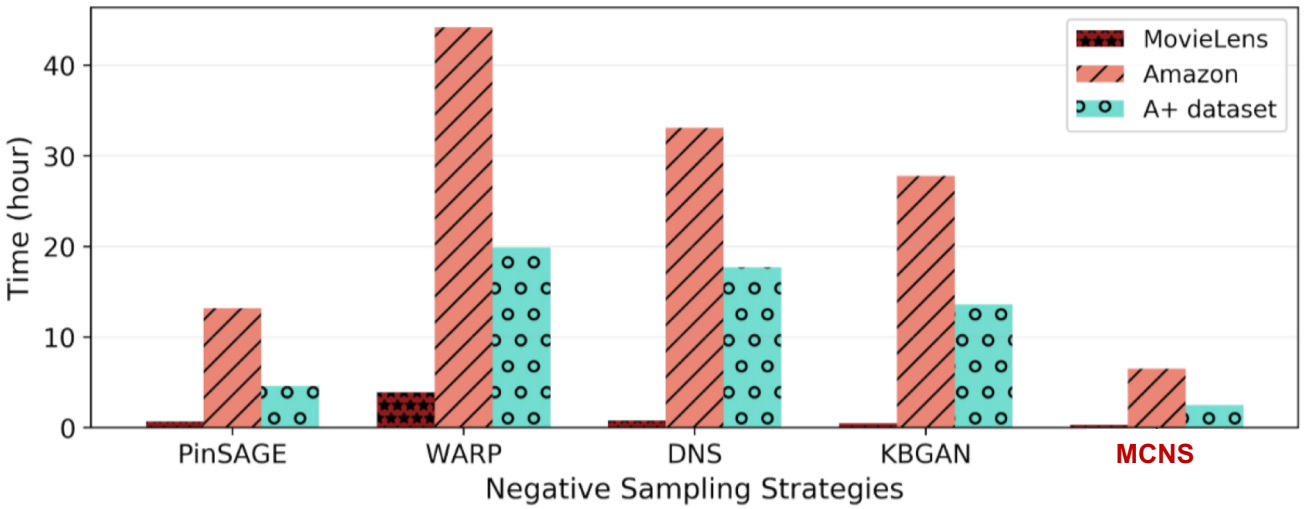
BlogCatalog: 10,312 nodes & 333,983 links

Evaluation Metric: Micro-F1

	Algorithm	DeepWalk			GCN			GraphSAGE		
	$T_R(\%)$	10	50	90	10	50	90	10	50	90
Micro -F1	$Deg^{0.75}$	31.6	36.6	39.1	36.1	41.8	44.6	35.9	42.1	44.0
	WRMF	30.9	35.8	37.5	34.2	41.4	43.3	34.4	41.0	43.1
	RNS	29.8	34.1	36.0	33.4	40.5	42.3	33.5	39.6	41.6
	PinSAGE	32.0	37.4	40.1	37.2	43.2	45.7	36.9	43.2	45.1
	WARP	35.1	40.3	42.1	39.9	45.8	47.7	40.1	45.5	47.5
	DNS	35.2	40.4	42.5	40.4	46.0	48.6	40.5	46.3	48.5
	IRGAN	34.3	39.6	41.8	39.1	45.2	47.9	38.9	45.0	47.6
	KBGAN	34.6	40.0	42.3	39.5	45.5	48.3	39.6	45.3	48.5
	MCNS	**36.1**	**41.2**	**43.3**	**41.7**	**47.3**	**49.9**	**41.6**	**47.5**	**50.1**

Efficiency Comparison

Runtime Comparisons:



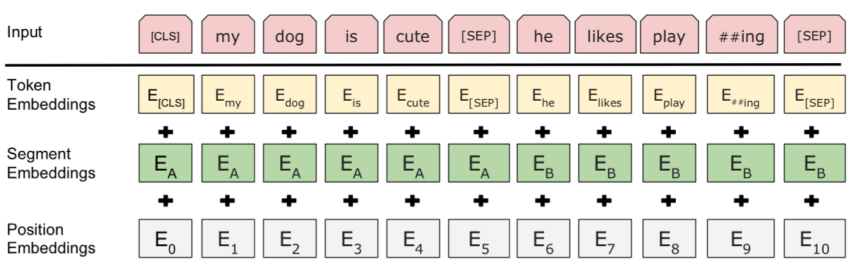
The runtime of MCNS and hard-samples or GAN-based strategies with GraphSAGE encoder in recommendation task.

Summary of MCNS

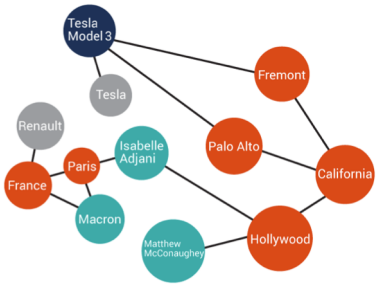
- Analyze the role of negative sampling from the perspectives of both objective and risk.
- Derive the theory and quantify that the negative sampling distribution should be positively but sub-linearly correlated to their positive sampling distribution.
- Approximate the positive distribution with self-contrast approximation and accelerating negative sampling by Metropolis-Hastings.
- Achieve start-of-art performance in recommendation, link prediction and node classification, on a total of 19 experimental settings.

Challenges

- How to combine Graph with Pre-Training (BERT/CL)?
- Graph Neural Network Pre-Training



+

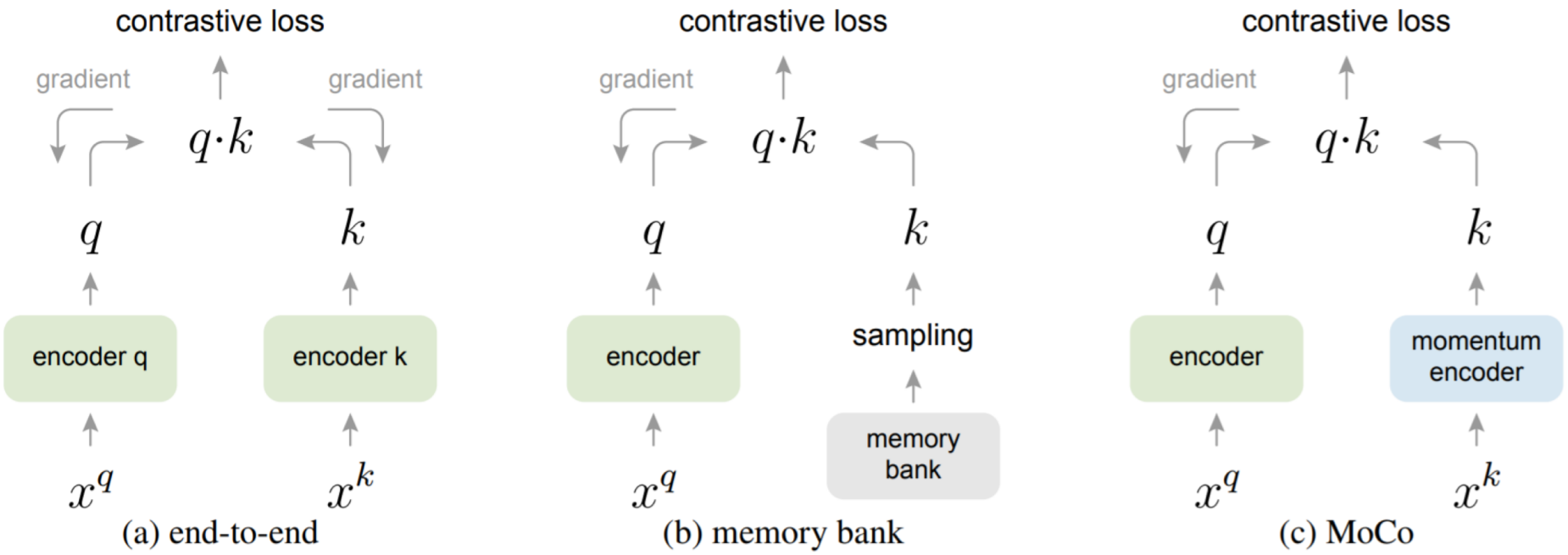


GCC: Graph Contrastive Coding for Graph Neural Network Pre-Training

- Momentum Contrast for Unsupervised Visual Representation Learning https://arxiv.org/pdf/1911.05722.pdf
- Train ResNet on ImageNet without image labels
  - Competitive results on ImageNet classification
  - Outperform supervised ResNet in 7 downstream tasks

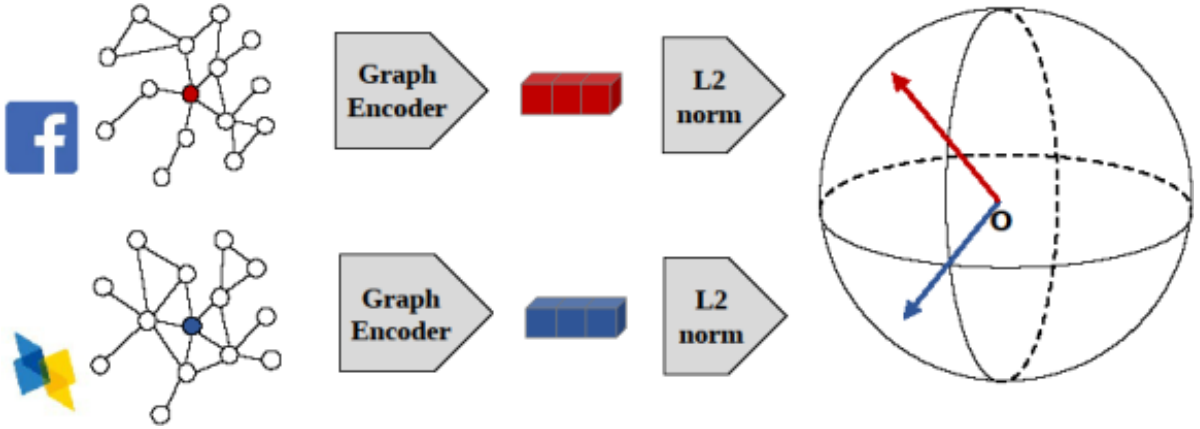
1. J. Qiu, Q. Chen, Y. Dong, J. Zhang, H. Yang, M. Ding, K. Wang, and J. Tang. GCC: Graph Contrastive Coding for Graph Neural Network Pre-Training. *KDD'20*.

Revisit MoCo



Graph Contrastive Coding (GCC)

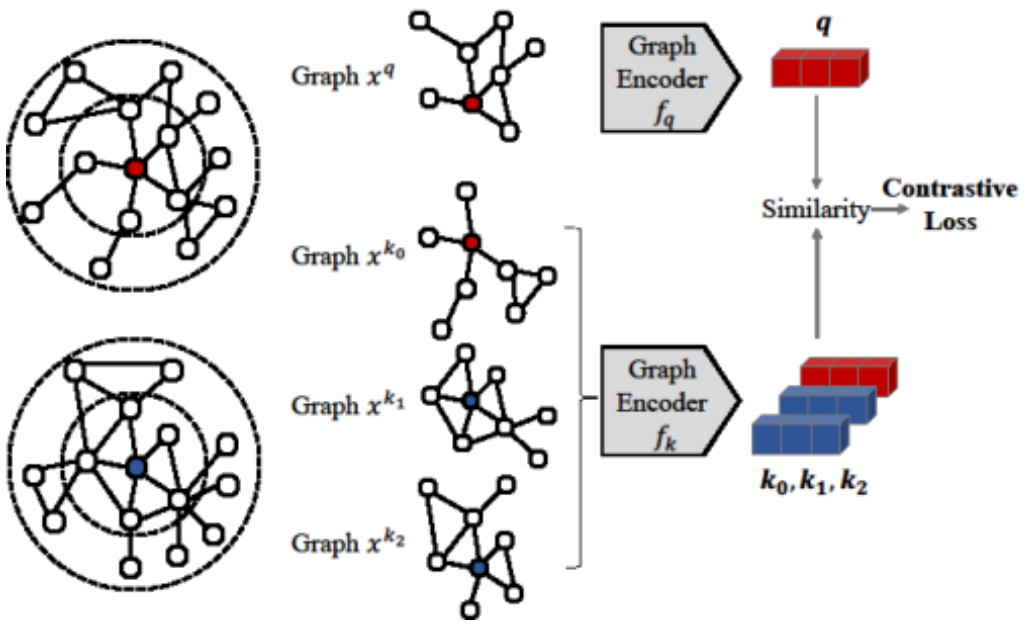
- Problem formulation:
  - Measuring vertex structural similarity



  - New wine in old bottles:
    - motif, centrality, clustering coefficient, structural diversity, edge density, etc

Graph Contrastive Coding (GCC)

- Define positive pairs by sub-graph sampling
  - Random walk with restart sub-graph sampling



1. J. Qiu, Q. Chen, Y. Dong, J. Zhang, H. Yang, M. Ding, K. Wang, and J. Tang. GCC: Graph Contrastive Coding for Graph Neural Network Pre-Training. *KDD'20*.

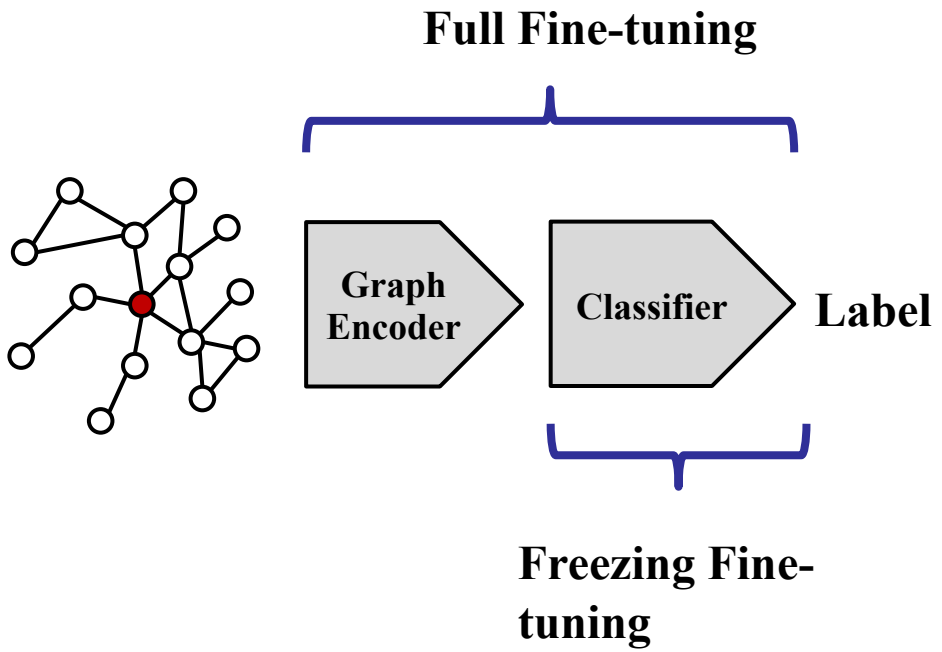
GCC Pre-Training / Fine-tuning

- Six real-world information networks for pre-training.

Table 1: Datasets for pre-training, sorted by number of vertices.

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-tuning Tasks:
  - Node classification
  - Graph classification
  - Similarity search
- Two fine-tuning settings.



Node Classification

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**

Graph Classification

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

Similarity Search

	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

Takeaway Messages

- Tackle non-robust, over-fitting, and over-smoothing
- Understanding GNNs as Signal Rescaling
  - Unifying GCNs
  - Unifying GAT
- Pre-training and Self-supervised learning is becoming more and more important

Related Publications

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

- 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.
- Yuxiao Dong, Ziniu Hu, Kuansan Wang, Yizhou Sun and Jie Tang. Heterogeneous Network Representation Learning. IJCAI'20.
- Ming Ding, Chang Zhou, Qibin Chen, Hongxia Yang, and Jie Tang. Cognitive Graph for Multi-Hop Reading Comprehension at Scale. ACL'19.
- Jie Zhang, Yuxiao Dong, Yan Wang, Jie Tang, and Ming Ding. ProNE: Fast and Scalable Network Representation Learning. IJCAI'19.
- 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.
- 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.
- 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, Jian Tang, Hao Ma, Yuxiao Dong, Kuansan Wang, and Jie Tang. DeepInf: Modeling Influence Locality in Large Social Networks. KDD'18.
- 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.
- Jie Tang, Jing Zhang, Limin Yao, Juanzi Li, Li Zhang, and Zhong Su. ArnetMiner: Extraction and Mining of Academic Social Networks. KDD'08.

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**)

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

Yuxiao Dong, Chi Wang, Hao Ma, Kuansan Wang (**Microsoft**)

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