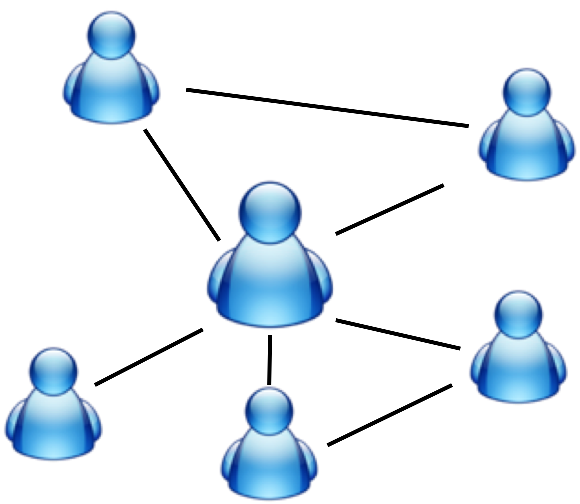
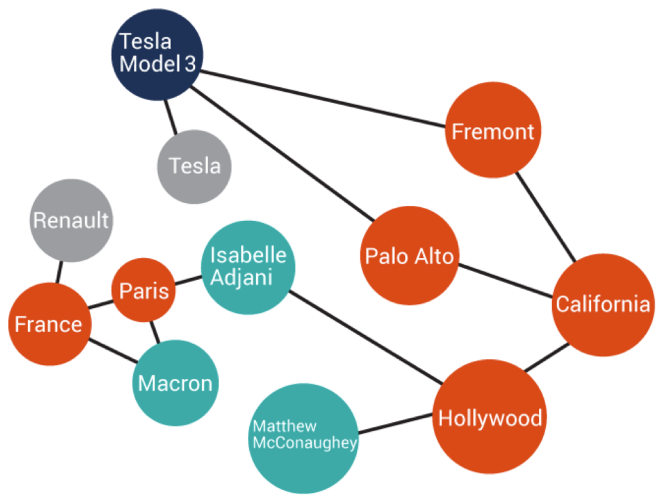
Network Embedding, Graph Neural Networks and Reasoning

Jie Tang
Computer Science
Tsinghua University

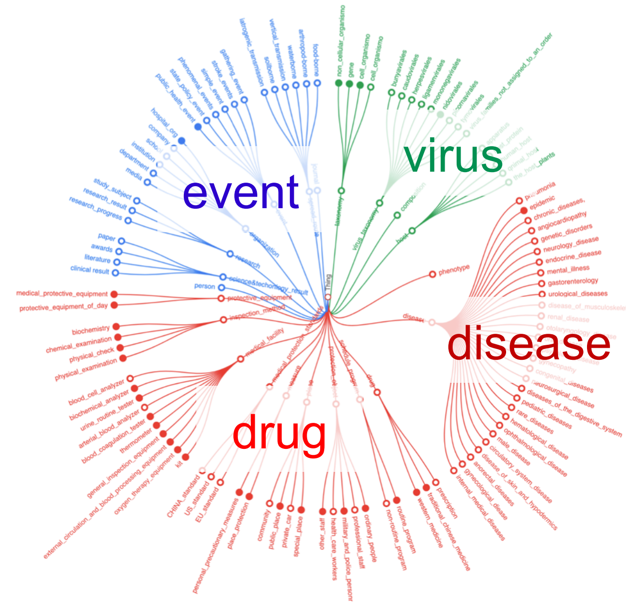
Networked World



Social Network



Knowledge Graph

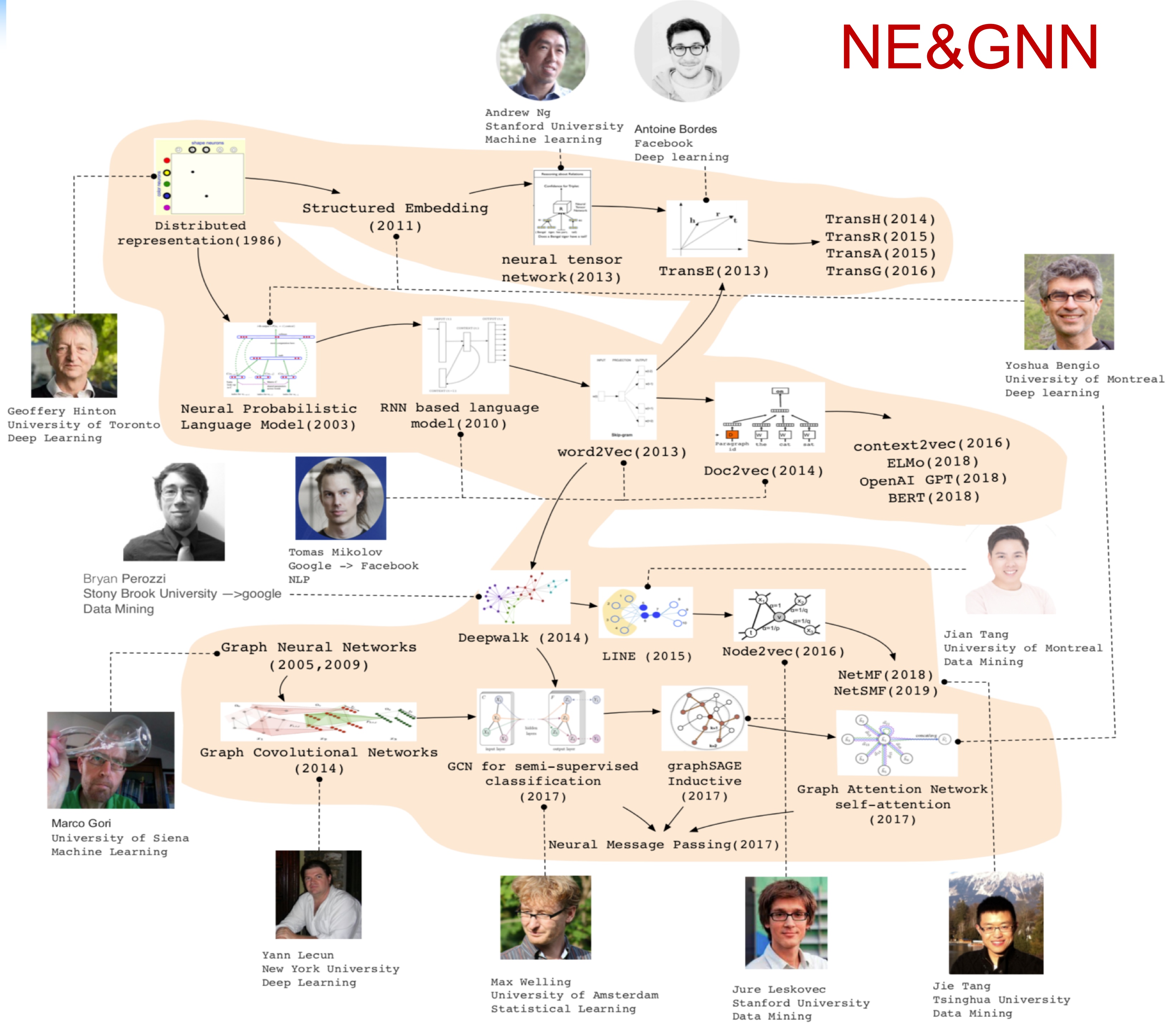


COVID Graph

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?

NE&GNN



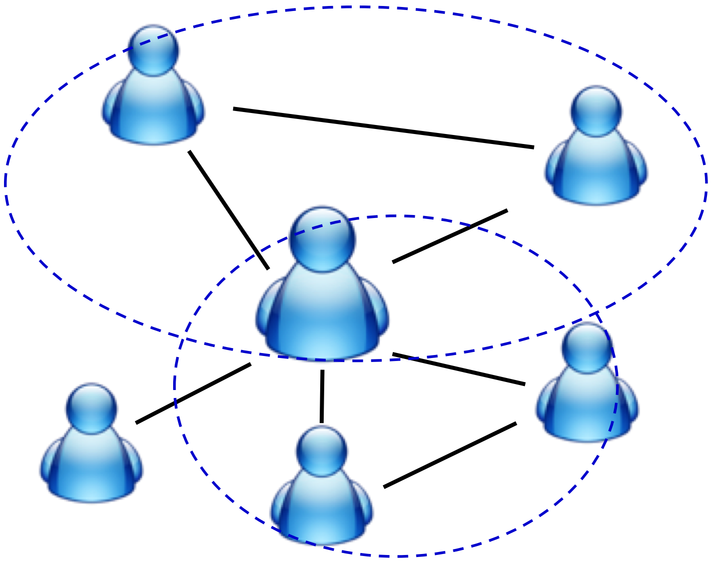
Andrew Ng
Stanford University
Machine learning
Antoine Bordes
Facebook
Deep learning
Distributed
representation(1986)
Structured Embedding
(2011)
neural tensor
network(2013)
TransE(2013)
TransH(2014)
TransR(2015)
TransA(2015)
TransG(2016)
Yoshua Bengio
University of Montreal
Deep learning
Geoffery Hinton
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)
Bryan Perozzi
Stony Brook University —>google
Data Mining
Tomas Mikolov
Google -> Facebook
NLP
Jian Tang
University of Montreal
Data Mining
Deepwalk (2014)
LINE (2015)
Node2vec(2016)
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)
Neural Message Passing(2017)
Marco Gori
University of Siena
Machine Learning
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

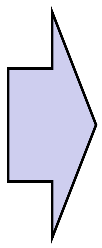
Agenda

- Network Embedding
- Revisiting Network Embedding
- Graph Neural Network
- Revisiting Graph Neural Network
- GNN&Reasoning
- Revisiting GNN&Reasoning

Representation Learning on Networks

Representation Learning/
Graph Embedding



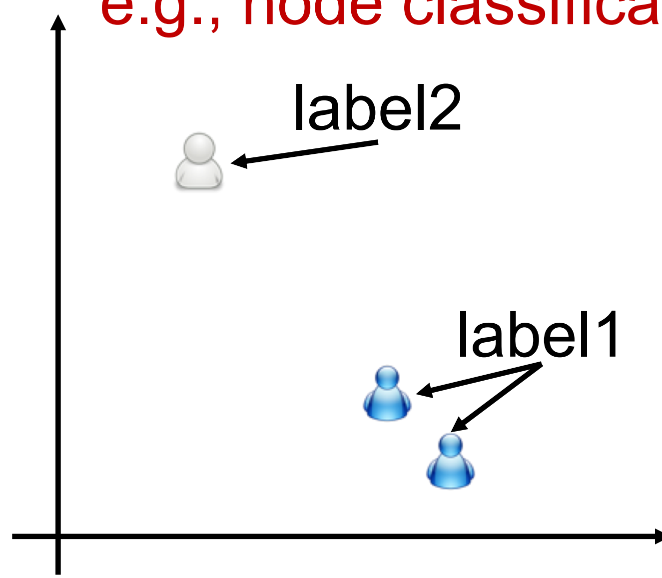


d-dimensional vector, $d \ll |V|$

0.8	0.2	0.3	…	0.0	0.0

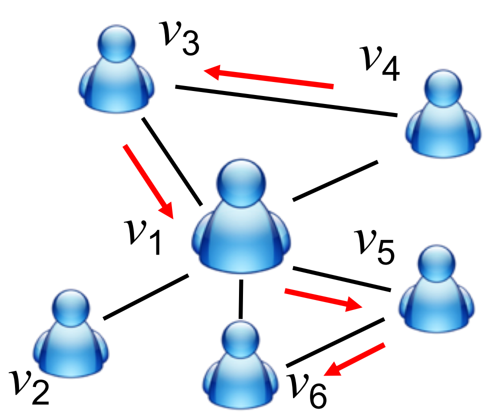
Users with the same label are located in closer

e.g., node classification

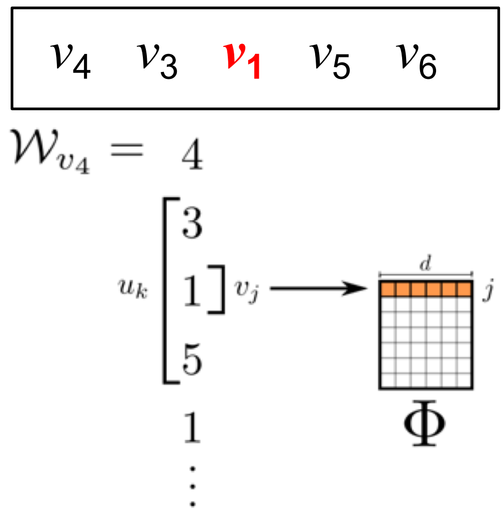


DeepWalk: Random Walk + Word2Vec

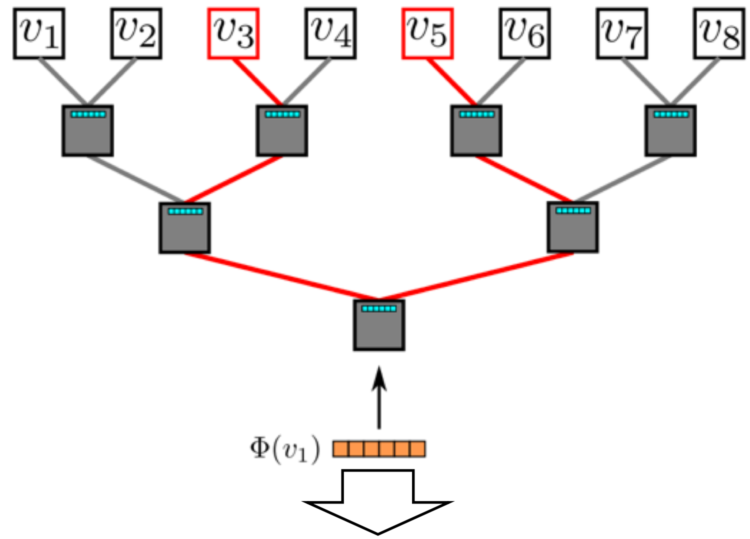
Random walk

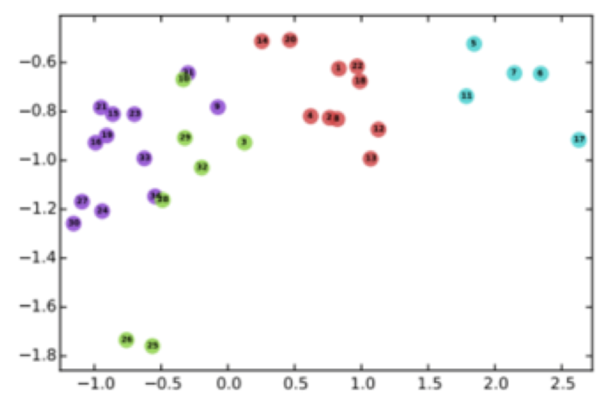


One example RW path



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

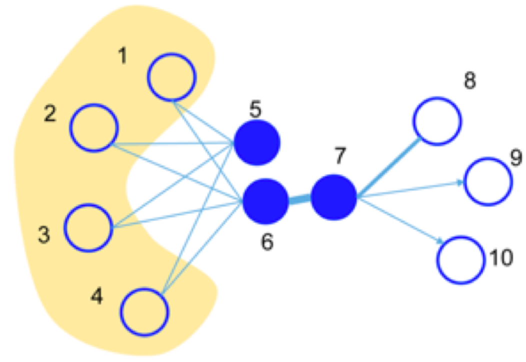
	% Labeled Nodes	1%	2%	3%	4%	5%	6%	7%	8%	9%	10%
Micro-F1(%)	DeepWalk	**37.95**	**39.28**	**40.08**	**40.78**	**41.32**	**41.72**	**42.12**	**42.48**	**42.78**	**43.05**
	SpectralClustering	—	—	—	—	—	—	—	—	—	—
	EdgeCluster	23.90	31.68	35.53	36.76	37.81	38.63	38.94	39.46	39.92	40.07
	Modularity	—	—	—	—	—	—	—	—	—	—
	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

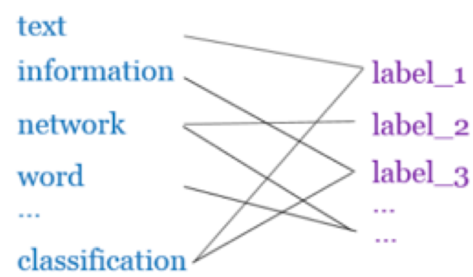
- DeepWalk utilizes fixed-length, unbiased random walks to generate context for each node, can we do better?

Later…

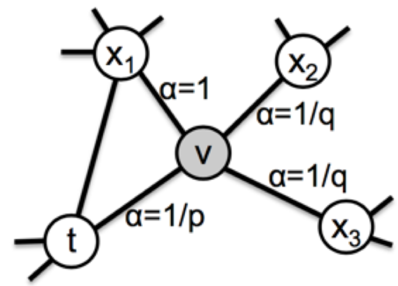
- LINE[1]: explicitly preserves both *first-order* and *second-order* proximities.



- 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

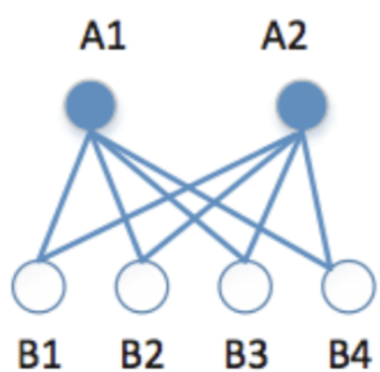
Definition

- 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

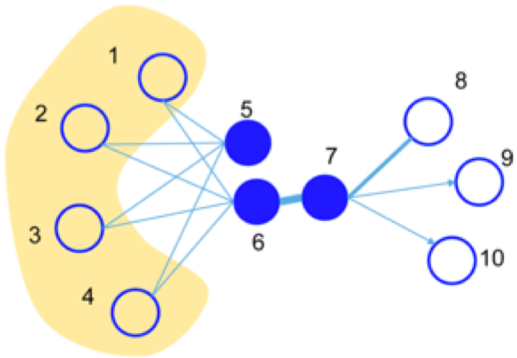
- Optimizing objective O_2 is computationally expensive.
- Adopt the approach of **negative sampling**. More specifically, it specifies the following objective function for each edge $e_{i,j}$,

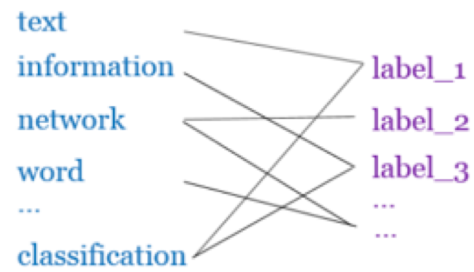
$$\underbrace{\log\sigma(\vec{u}_j'^{T} \cdot \vec{u}_i)}_{\text{the observed edges}} + \underbrace{\sum_{i=1}^{K} E_{v_n} \sim P_n(v)[\log\sigma(-\vec{u}_n'^{T} \cdot \vec{u}_i)]}_{\text{the negative edges drawn from the noise distribution}}$$

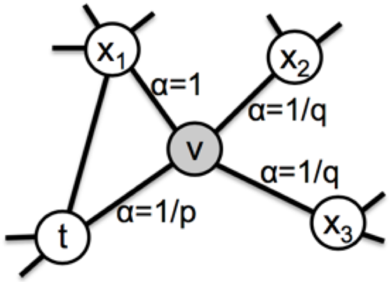
- We can use **asynchronous stochastic gradient algorithm (ASGD)** for optimizing above Eqn.

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.



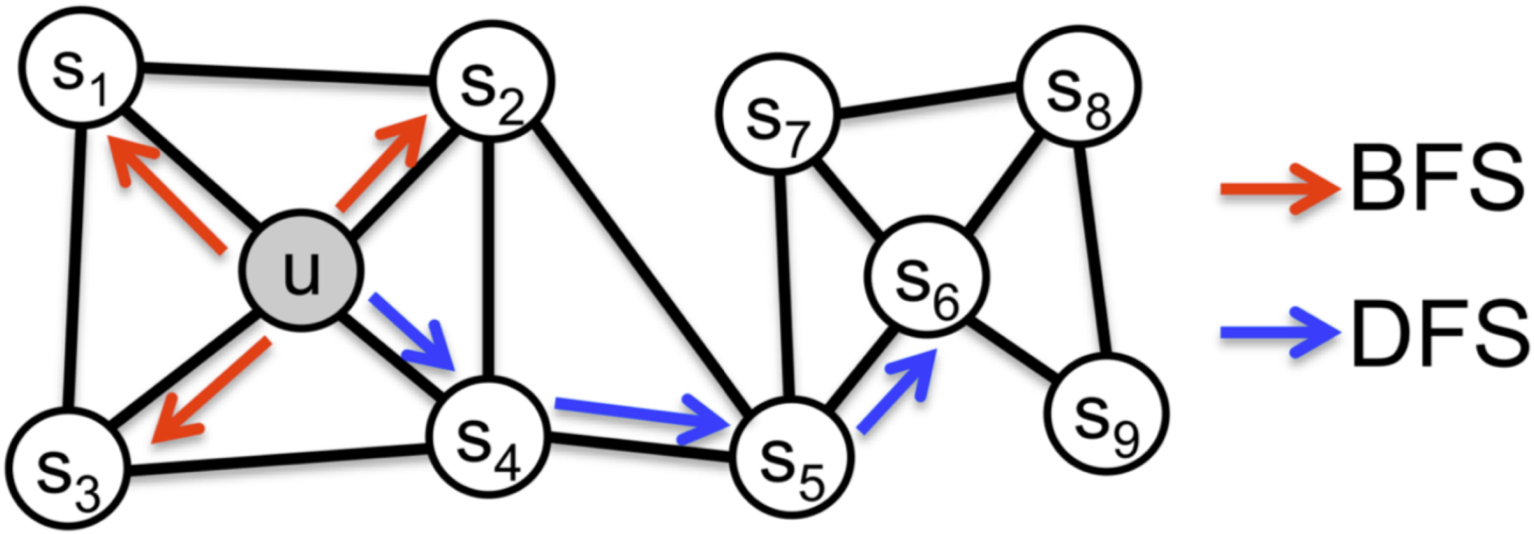




1. J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei. 2015. Line: Large-scale information network embedding. *WWW'15*, 1067–1077.
2. J. Tang, M. Qu, and Q. Mei. 2015. Pte: Predictive text embedding through large-scale heterogeneous text networks. *KDD'15*, 1165–1174.
3. A. Grover and J. Leskovec. 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

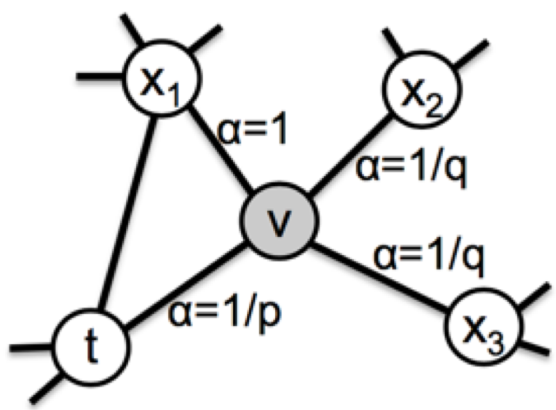


- Biased walks is a special case of random walk, thus node2vec is a special case of DeepWalk

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

node2vec

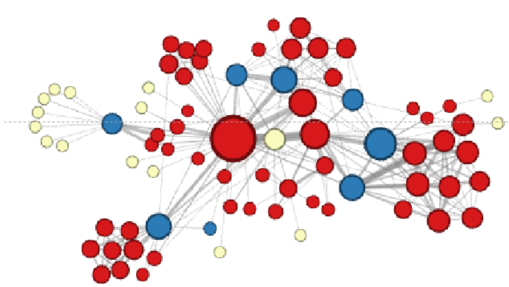
- Biased random walk R that given a node v generates random walk neighborhood $N_{rw}(v)$
- Return parameter p:
  - o Return back to the previous node
- In-out parameter q:
  - o Moving outwards (DFS) vs. inwards (BFS)



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:



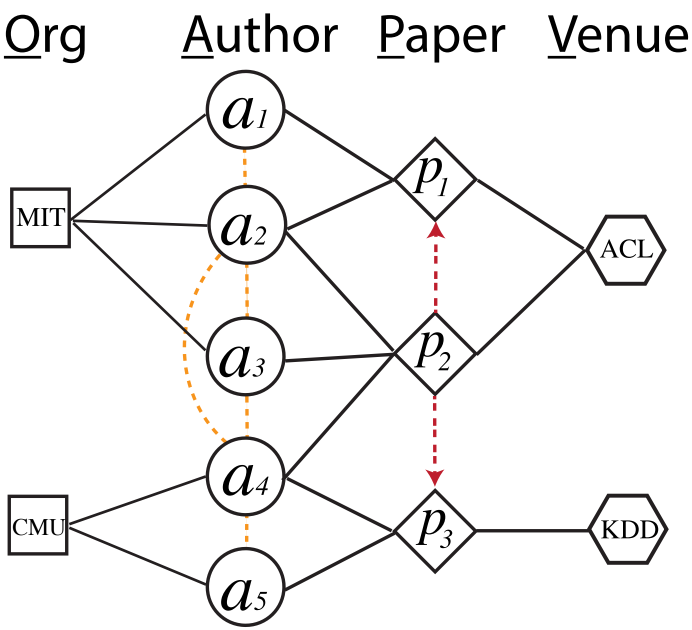
p=1, q=2
Microscopic view of the network neighbourhood

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

Picture snipped from Leskovec

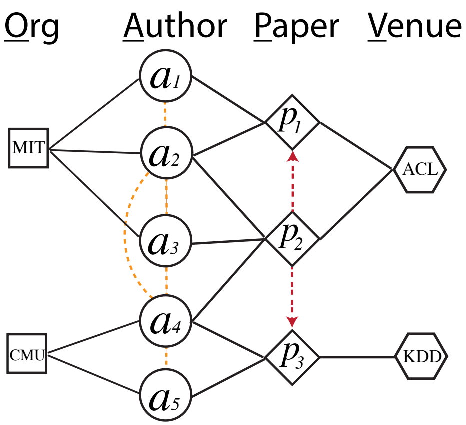
Metapath2vec: Heterogeneous Random Walk

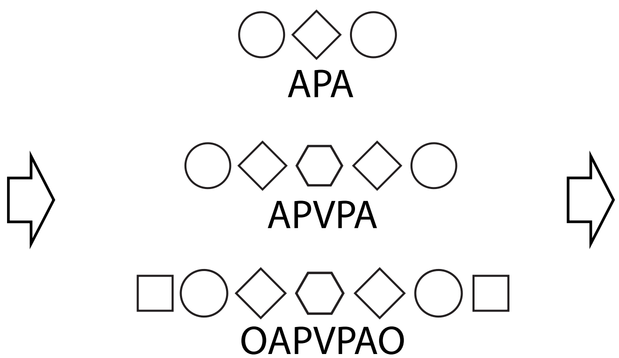
- Input: a heterogeneous graph $G = (V, E)$
- Output: $\boldsymbol{X} \in R^{|V| \times k}$, $k \ll |V|$, k-dim vector $\boldsymbol{X}_v$ for each node *v*.

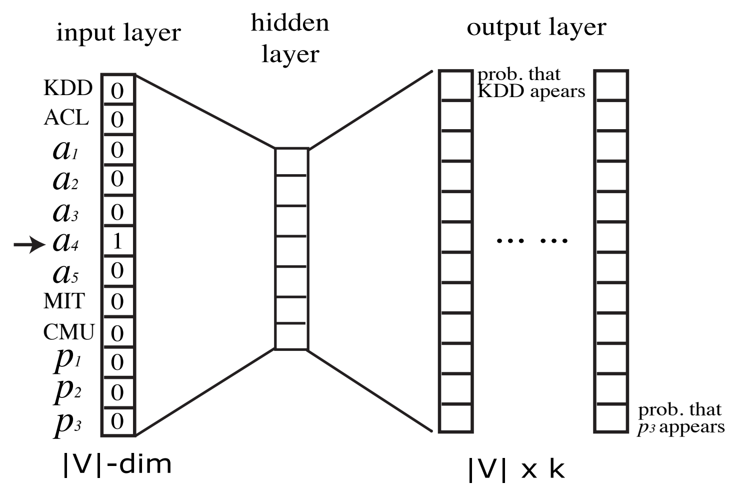


- How do we random walk over heterogeneous networks?
- How do we apply skip-gram over different types of nodes?

Metapath2vec: Heterogeneous Random Walk





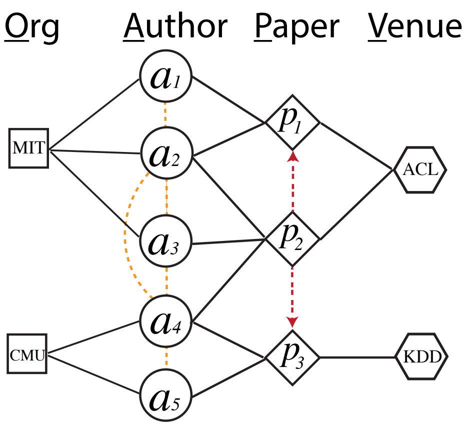


meta-path-based random walks

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

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

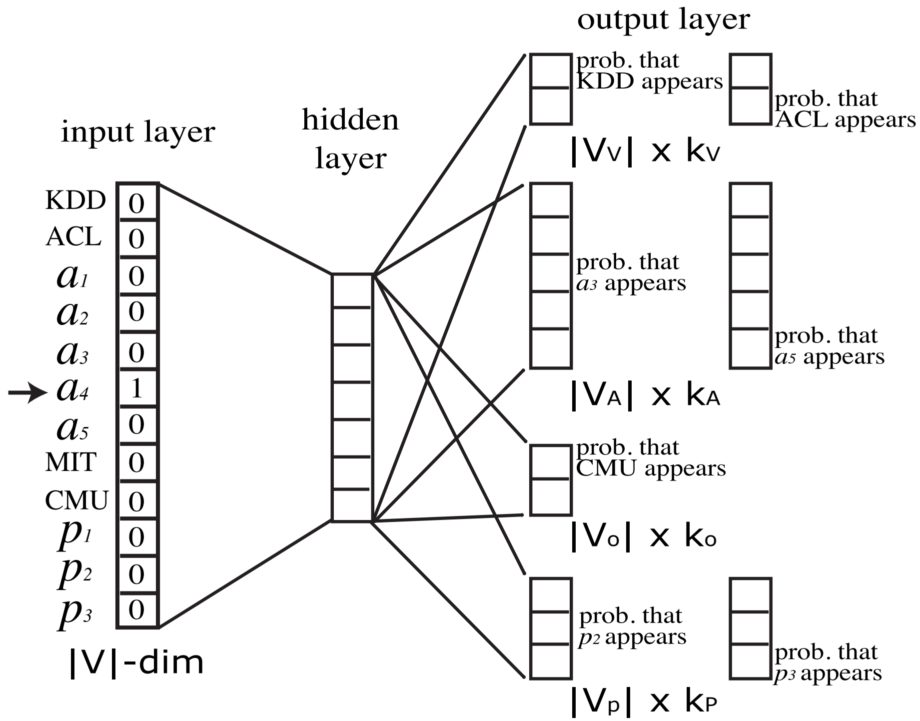
- The transition probability at step *i* is defined as

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

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

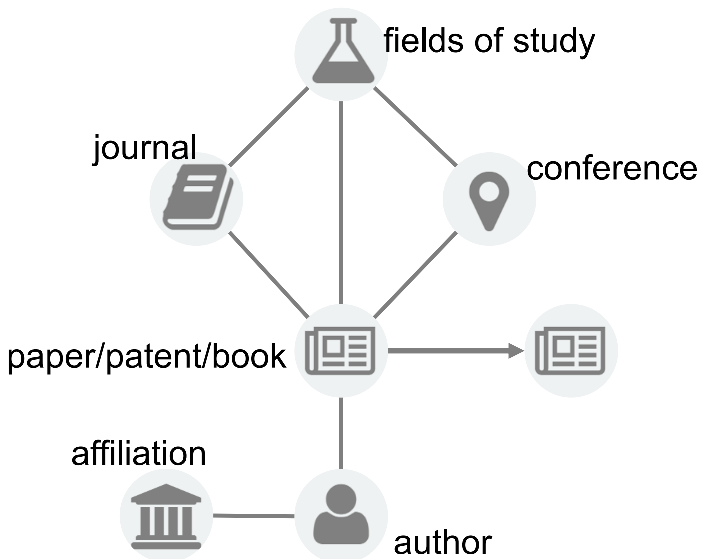
$$p(v^{i+1}|v_t^i) = p(v^{i+1}|v_1^i), \text{ if } t = l$$

Metapath2vec: Heterogeneous Random Walk

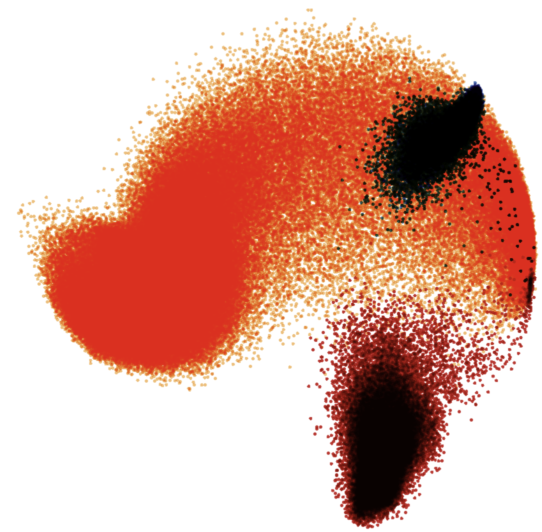


$$\mathcal{O}(\mathbf{X}) = \log \sigma(X_{c_t} \cdot X_v) + \sum_{k=1}^{K} \mathbb{E}_{u_t^k \sim P_t(u_t)} [\log \sigma(-X_{u_t^k} \cdot X_v)]$$

Application: Embedding Academic Graph

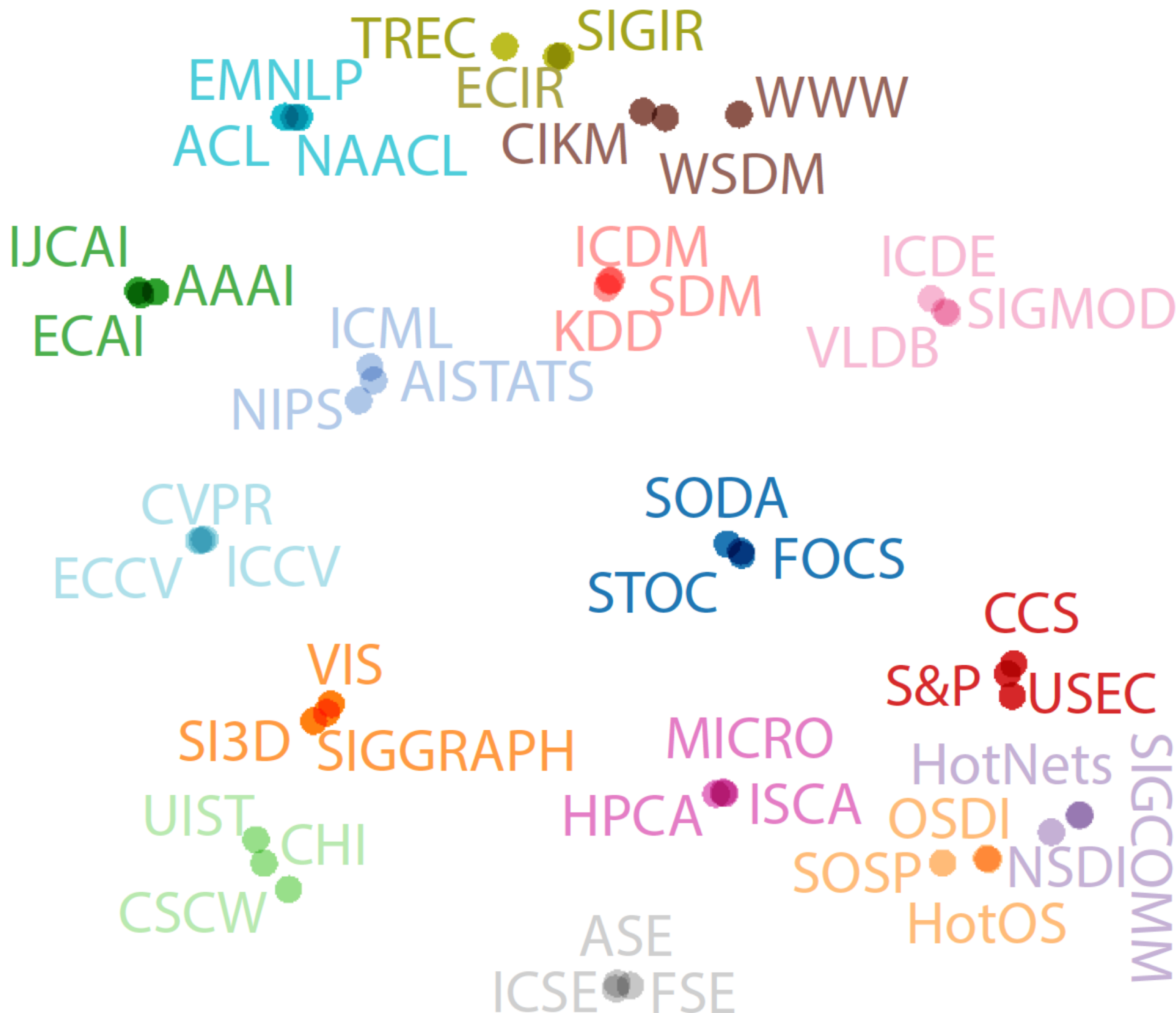


⇨ metapath2vec / metapath2vec++ ⇨

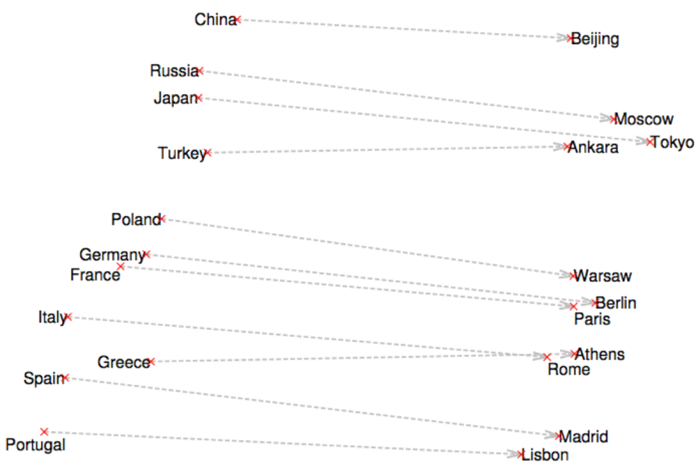


Microsoft Academic Graph
&
AMiner

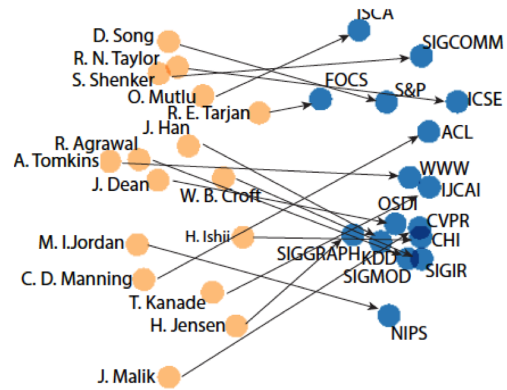
Application 2: Node Clustering



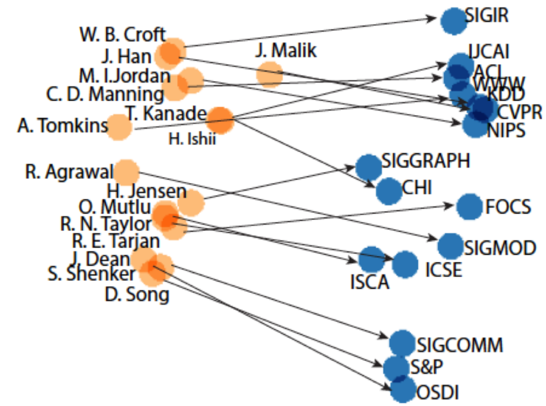
Visualization



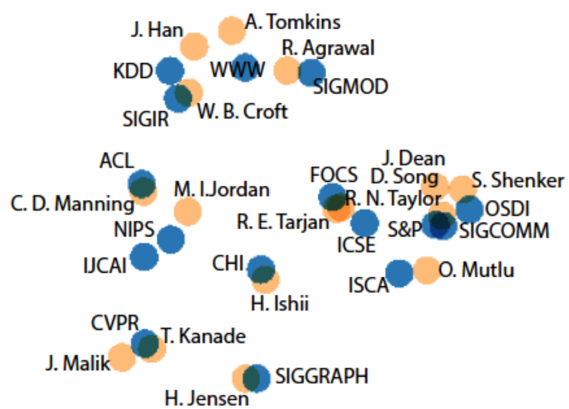
word2vec [Mikolov, 2013]



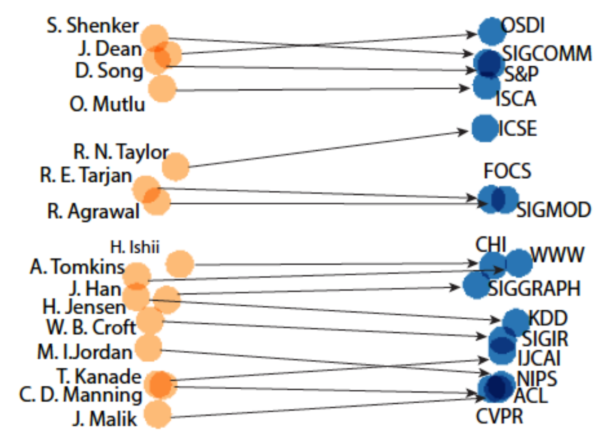
(a) DeepWalk/node2vec



(b) PTE



(c) *metapath2vec*

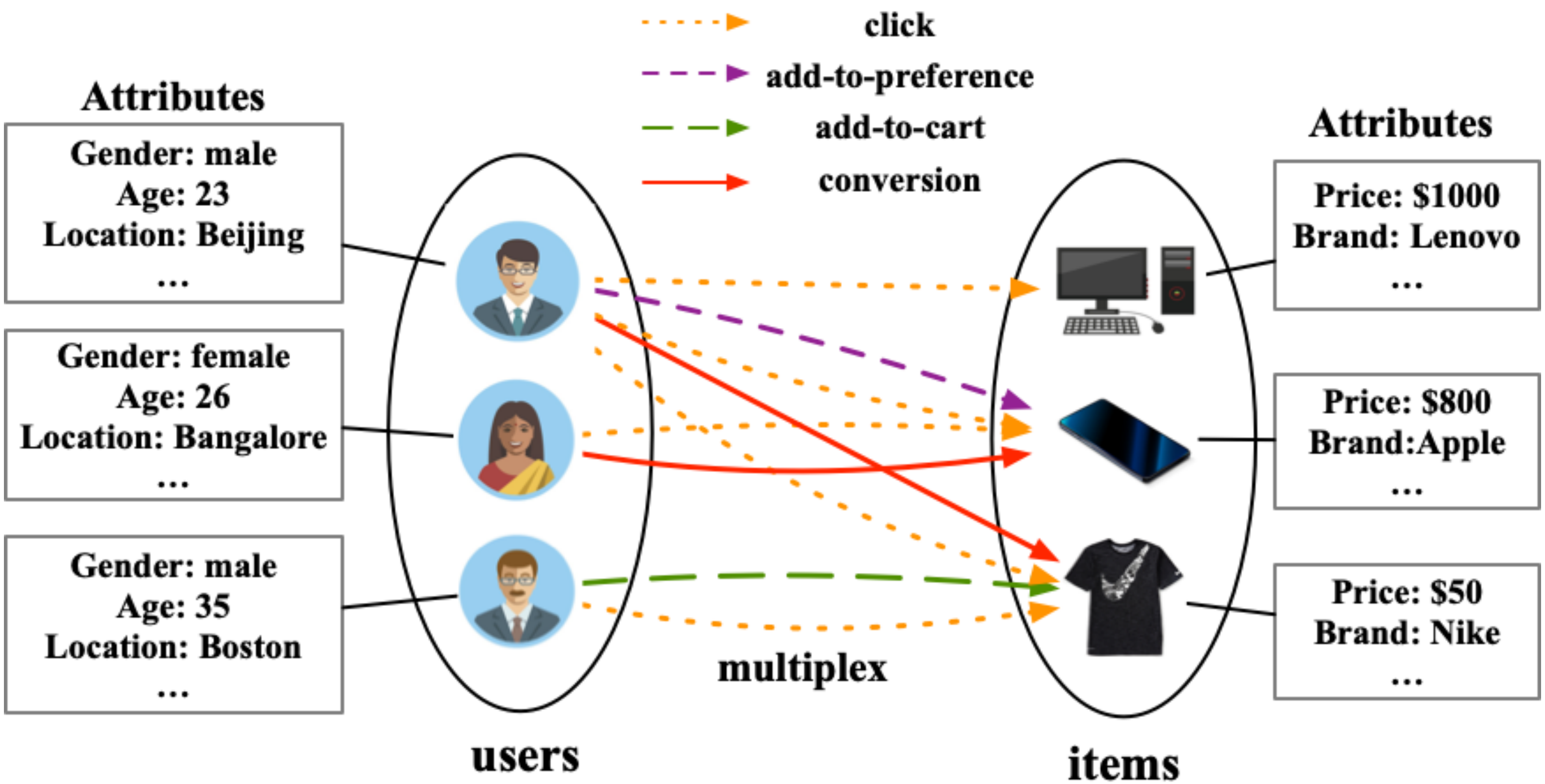


(d) *metapath2vec++*

http://projector.tensorflow.org/

GATNE: Attributed Multiplex Heterogeneous Network Embedding

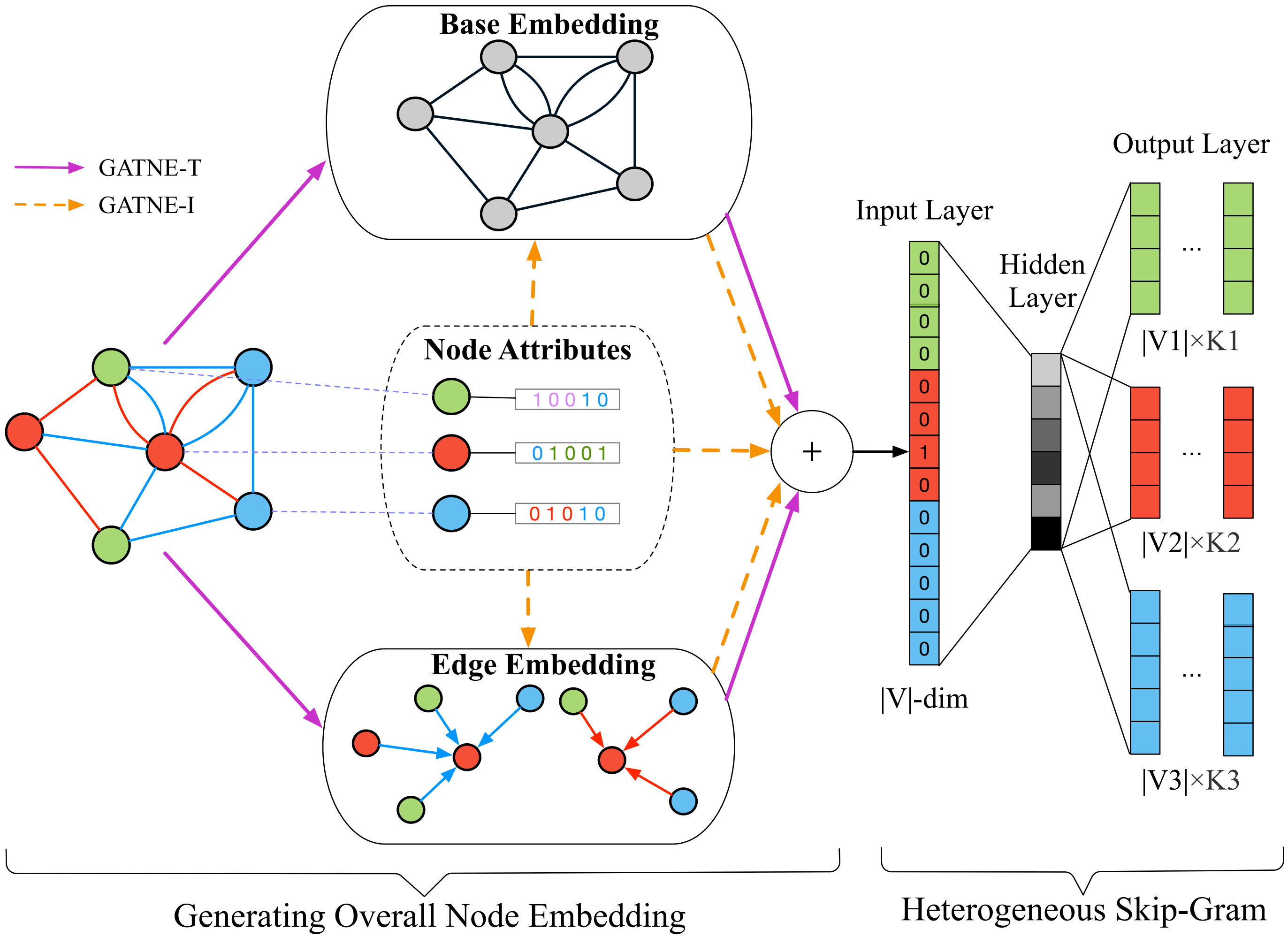
- Recommend items to users by considering *Attributed Multiplex Heterogeneous Networks* (AMHEN)



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.

Agenda

- Network Embedding
- Revisiting Network Embedding
- Graph Neural Network
- Revisiting Graph Neural Network
- GNN&Reasoning
- Revisiting GNN&Reasoning

Questions

- What are the fundamentals underlying the different models?

or

- Can we unify the different graph embedding approaches?

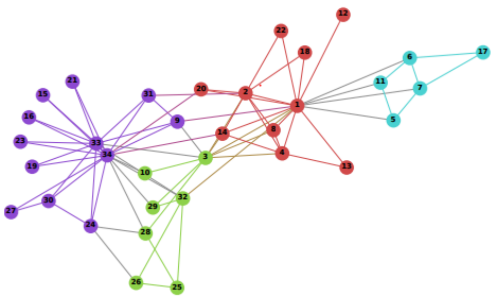
Unifying DeepWalk, LINE, PTE, and node2vec into Matrix Forms

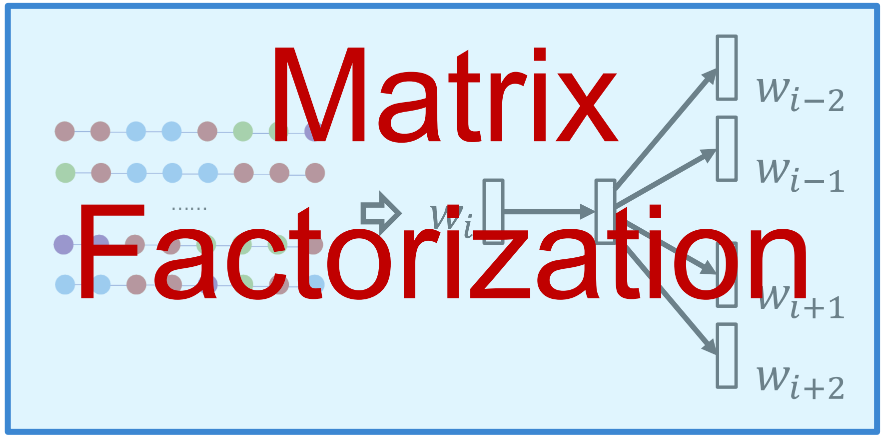
Algorithm	Closed Matrix Form
DeepWalk	$\log\left(\mathrm{vol}(G)\left(\frac{1}{T}\sum_{r=1}^{T}(\boldsymbol{D}^{-1}\boldsymbol{A})^r\right)\boldsymbol{D}^{-1}\right)-\log b$
LINE	$\log\left(\mathrm{vol}(G)\boldsymbol{D}^{-1}\boldsymbol{A}\boldsymbol{D}^{-1}\right)-\log b$
PTE	$\log\left(\begin{bmatrix}\alpha\,\mathrm{vol}(G_{\mathrm{ww}})(\boldsymbol{D}_{\mathrm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

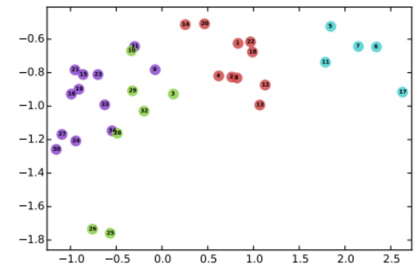
$\boldsymbol{A}$: $\boldsymbol{A}\in\mathbb{R}_+^{|V|\times|V|}$ is G's adjacency matrix with $\boldsymbol{A}_{i,j}$ as the edge weight between vertices i and j;
$\boldsymbol{D}_{\mathrm{col}}$: $\boldsymbol{D}_{\mathrm{col}}=\mathrm{diag}(\boldsymbol{A}^\top\boldsymbol{e})$ is the diagonal matrix with 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.* **The most cited paper in WSDM'18 as of May 2019**

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

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

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

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

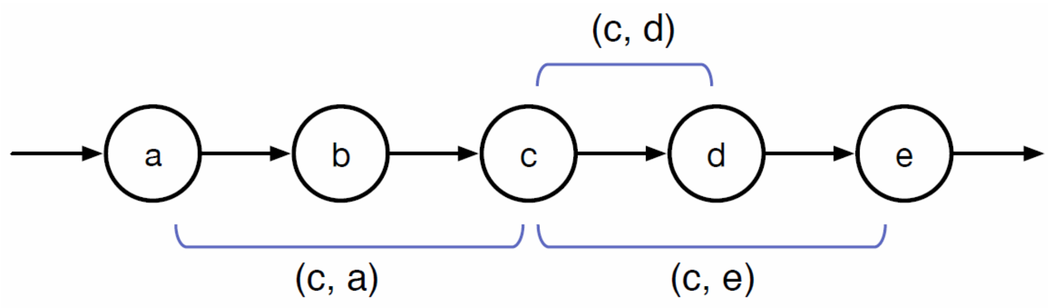
Levy and Goldberg. Neural word embeddings as implicit matrix factorization. In *NIPS 2014*

Understanding random walk + skip gram

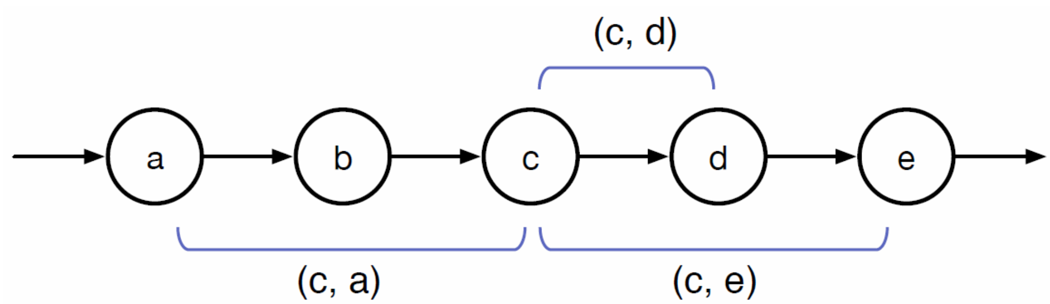
for $n = 1, 2, \ldots, N$ **do**
- Pick w_1^n according to a probability distribution $P(w_1)$;
- Generate a vertex sequence $(w_1^n, \cdots, w_L^n)$ of length L by a random walk on network G;
- **for** $j = 1, 2, \ldots, L - T$ **do**
  - **for** $r = 1, \ldots, T$ **do**
    - Add vertex-context pair (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

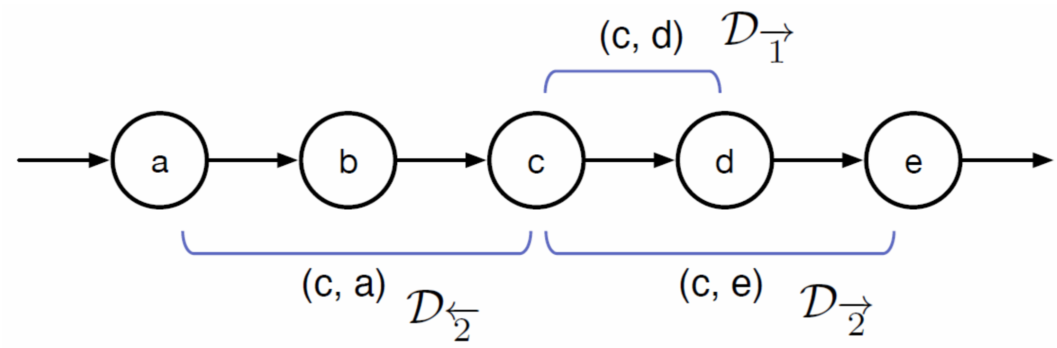


Understanding random walk + skip gram



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 $L\rightarrow\infty$

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

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

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

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

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

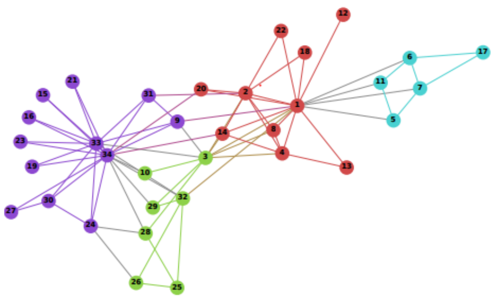
$$\frac{\#(w,c)\,|\mathcal{D}|}{\#(w)\cdot\#(c)} \xrightarrow{p} \frac{\mathrm{vol}(G)}{2T}\left(\frac{1}{d_c}\sum_{r=1}^{T}(\boldsymbol{P}^r)_{w,c} + \frac{1}{d_w}\sum_{r=1}^{T}(\boldsymbol{P}^r)_{c,w}\right)$$

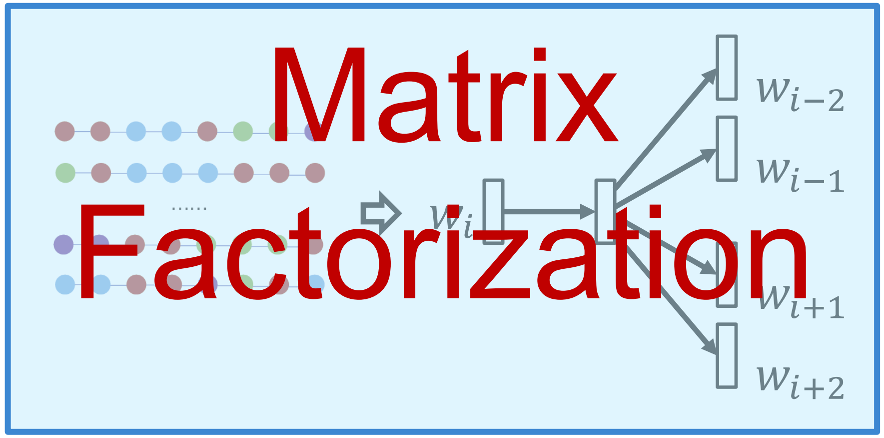
$$\frac{\mathrm{vol}(G)}{2T}\left(\sum_{r=1}^{T}\boldsymbol{P}^{r}\boldsymbol{D}^{-1} + \sum_{r=1}^{T}\boldsymbol{D}^{-1}(\boldsymbol{P}^{r})^{\top}\right)$$

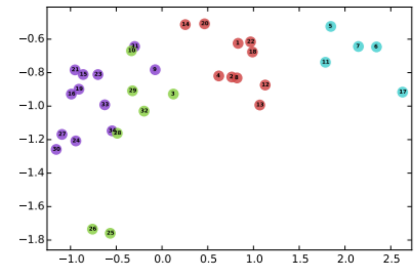
$$=\frac{\mathrm{vol}(G)}{2T}\left(\sum_{r=1}^{T}\underbrace{\boldsymbol{D}^{-1}\boldsymbol{A}\times\cdots\times\boldsymbol{D}^{-1}\boldsymbol{A}}_{r\text{ terms}}\boldsymbol{D}^{-1} + \sum_{r=1}^{T}\boldsymbol{D}^{-1}\underbrace{\boldsymbol{A}\boldsymbol{D}^{-1}\times\cdots\times\boldsymbol{A}\boldsymbol{D}^{-1}}_{r\text{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

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

DeepWalk is factorizing a matrix







DeepWalk is asymptotically and implicitly factorizing

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

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

$\boldsymbol{A}$ 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

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

- Recall DeepWalk's matrix form:

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

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

PTE

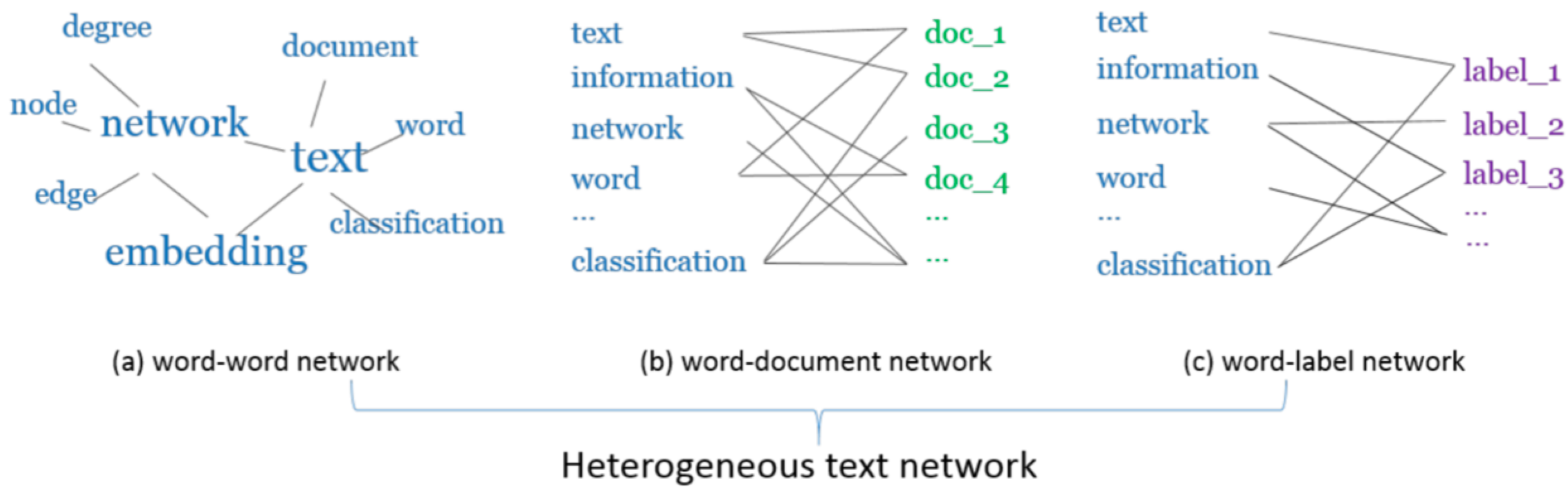


Figure 2: Heterogeneous Text Network.

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

Understanding node2vec

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

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

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

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

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

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

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

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

$\boldsymbol{z}_v^{\top}\boldsymbol{z}_c$ → 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

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

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

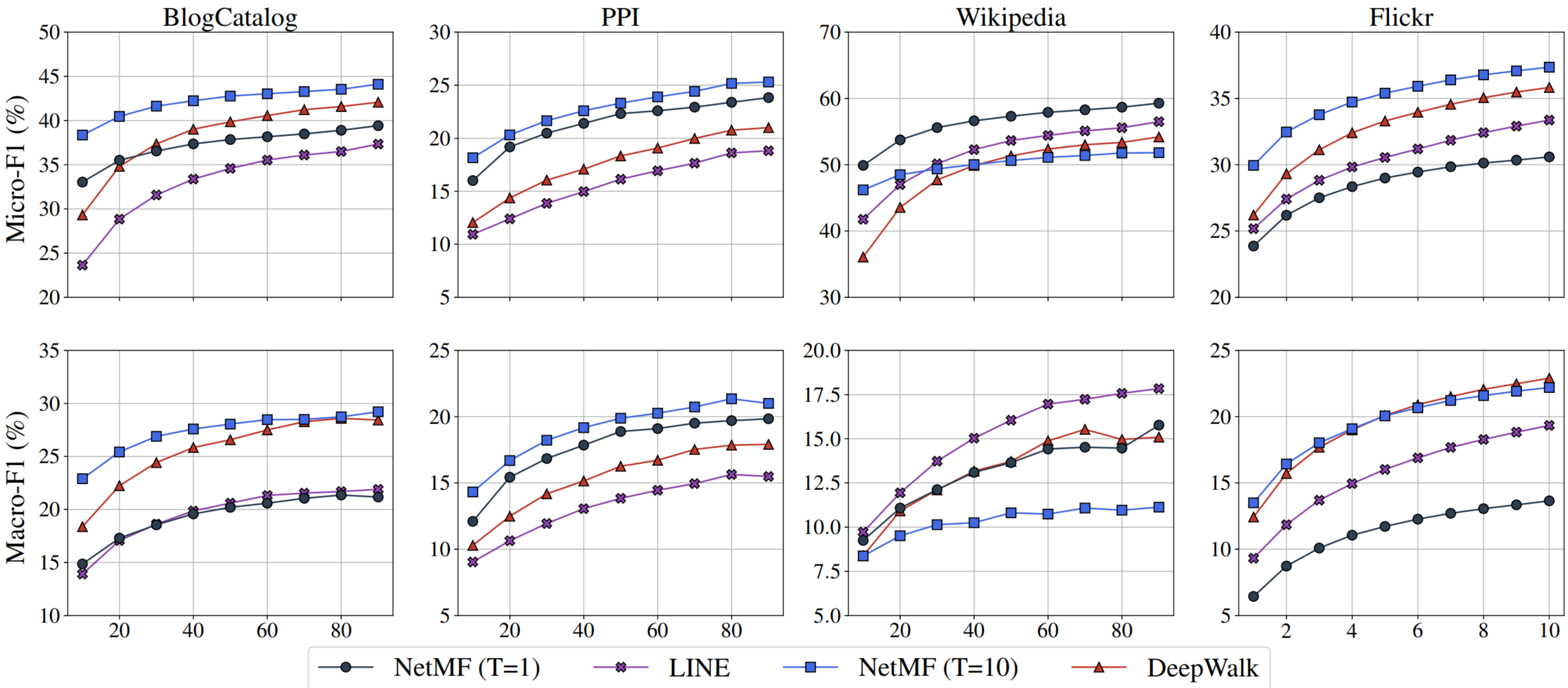
- and because $M'_{i,j}=\max(M_{i,j}, 1)>=1$, we have

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

- 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;

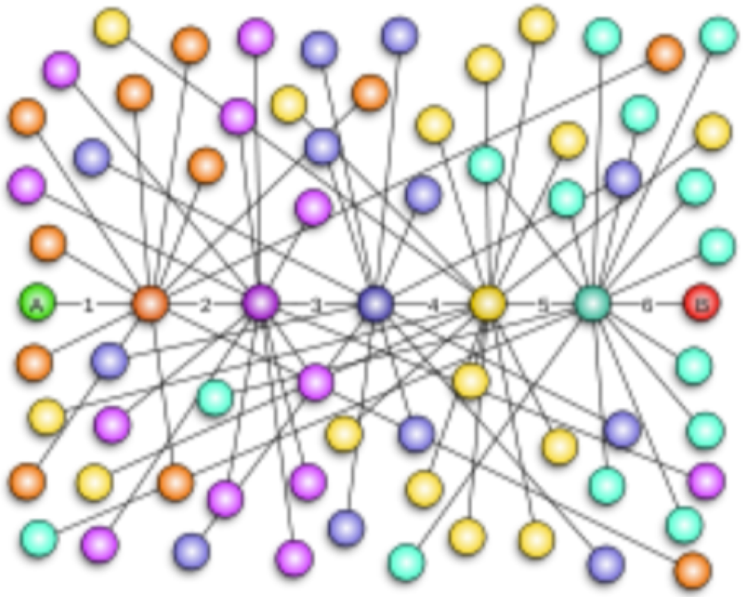
The *x*-axis represents the ratio of labeled data (%)

Network Embedding as Matrix Factorization

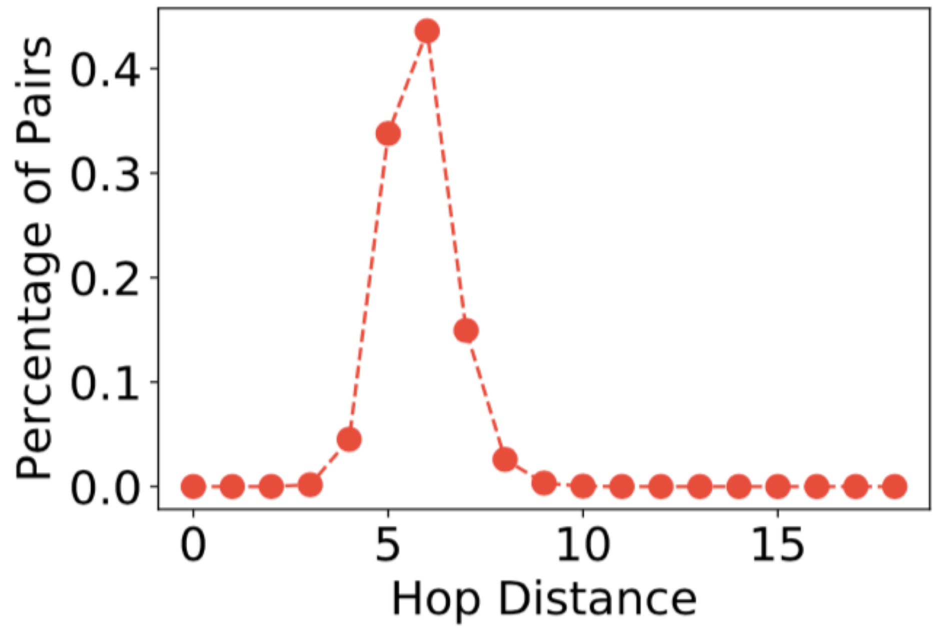
- DeepWalk $$\log\left(\frac{\mathrm{vol}(G)}{b}\left(\frac{1}{T}\sum_{r=1}^{T}\left(\boldsymbol{D}^{-1}\boldsymbol{A}\right)^{r}\right)\boldsymbol{D}^{-1}\right)$$
- LINE $$\log\left(\frac{\mathrm{vol}(G)}{b}\boldsymbol{D}^{-1}\boldsymbol{A}\boldsymbol{D}^{-1}\right)$$
- PTE $$\log\left(\begin{bmatrix}\alpha\,\mathrm{vol}(G_{\mathsf{ww}})(\boldsymbol{D}^{\maths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

1. Qiu et al. Network embedding as matrix factorization: unifying deepwalk, line, pte, and node2vec. In *WSDM'18*. **The most cited paper in WSDM'18 as of Aug 2018**

Challenge in NetMF



Small world



Academic graph

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

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

- 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

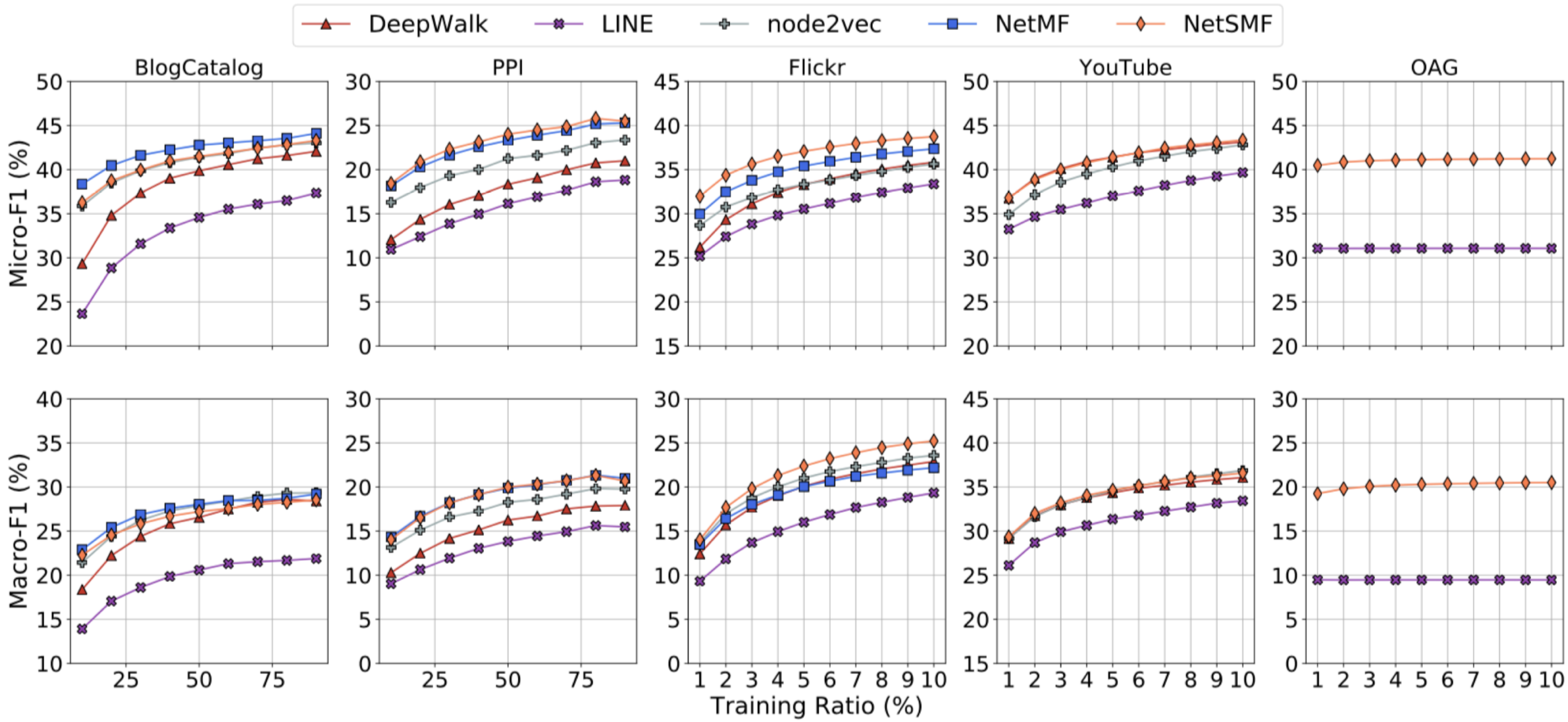


Figure 7: Predictive performance on varying the ratio of training data. The x-axis represents the ratio of labeled data (%), and the y-axis in the top and bottom rows denote the Micro-F1 and Macro-F1 scores

** 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 $(r_i = c_j, r_i^T c_j \rightarrow \infty, s.t. \hat{p} \rightarrow 1)$
- 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

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

- Matrix to be factorized (sparse)

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

NE as Sparse Matrix Factorization

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

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

- 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

Higher-order Cheeger's inequality

- $L = U \Lambda U^{-1}$, where $\Lambda = diag([\lambda_1, \dots, \lambda_n])$ with $0 = \lambda_1 \leq \cdots \leq \lambda_n$
- Bridge graph spectrum and graph partitioning

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

- *k*-way Cheeger constant $\rho_G(k)$: reflects the effect of the graph partitioned into *k* parts. A smaller value of $\rho_G(k)$ means a better partitioning effect.

1. J. R. Lee, S. O. Gharan, L. Trevisan. Multiway spectral partitioning and higher-order cheeger inequalities. Journal of the ACM (JACM), 2014, 61(6): 37.

Higher-order Cheeger's inequality

- Bridge graph spectrum and graph partition

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

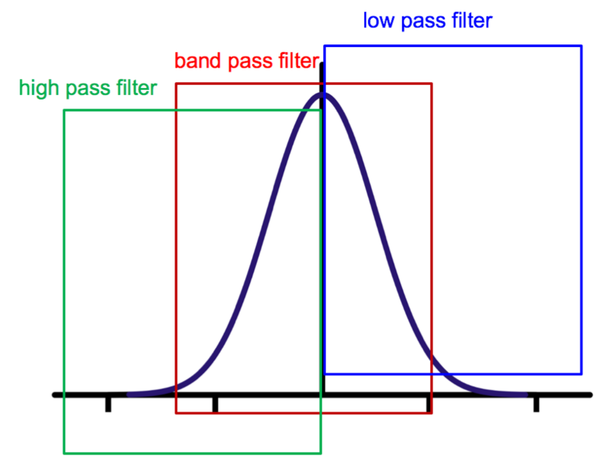
- low eigenvalues control the global information
- high eigenvalues control the local information
- example: $\lambda_2 = 0 \Leftrightarrow$ Graph is disconnected.
- spectral propagation: propagate the initial network embedding in the spectrally modulated network !

NE Enhancement via Spectral Propagation

- the form of the spectral filter

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

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



- Band-pass (low-pass, high-pass)

- pass eigenvalues within a certain range and weaken eigenvalues outside that range

- amplify local and global network information

Chebyshev Expansion for Efficiency

- Utilize truncated Chebyshev expansion to avoid explicit eigendecomposition and Fourier transform

$$\widetilde{L} \approx U \sum_{i=0}^{k-1} c_i(\theta) T_i(\bar{\Lambda}) U^T = \sum_{i=0}^{k-1} c_i(\theta) T_i(\bar{L})$$

$$\bar{\Lambda} = -\tfrac{1}{2}[(\Lambda - \mu E_n)^2 - E_n],\ \bar{L} = -\tfrac{1}{2}[(L - \mu E_n)^2 - E_n].\ \bar{\lambda} = \tfrac{1}{2}[(\lambda - \mu)^2 - 1] \in [-1, 1],$$

- Calculate the coefficient of Chebyshev expansion

$$c_i(\theta) = \begin{cases} \dfrac{1}{\pi}\displaystyle\int_{-1}^{1} \dfrac{T_i(x)e^{-x\theta}}{\sqrt{1-x^2}}dx = (-)^i I_i(\theta) & , i = 0 \\ \dfrac{2}{\pi}\displaystyle\int_{-1}^{1} \dfrac{T_i(x)e^{-x\theta}}{\sqrt{1-x^2}}dx = 2(-)^i I_i(\theta) & , i \neq 0 \end{cases}$$

where $I_i(\theta)$ is the modified Bessel function of the first kind

Chebyshev Expansion for Efficiency

- NE

$$R_d \leftarrow D^{-1}A(E_n - \widetilde{L})R_d$$

$$= D^{-1}A\{E_n - [I_0(\theta)T_0(\bar{L}) + 2\sum_{i=1}^{k-1}(-)^i I_i(\theta)T_i(\bar{L})]\}R_d$$

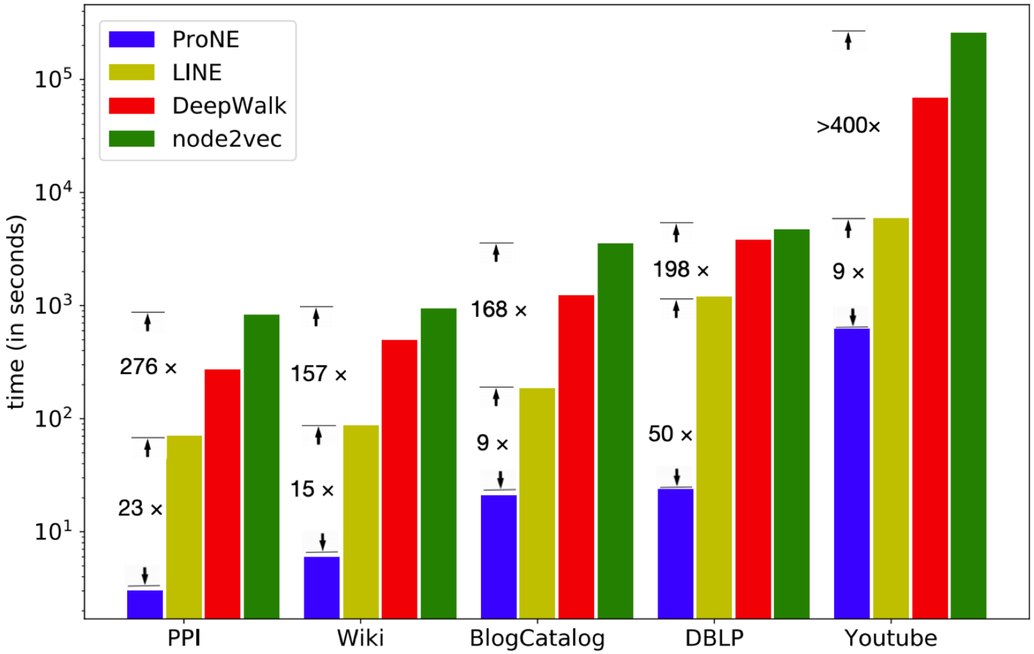
- Denote $\bar{R}_d^{(i)} = T_i(\bar{L})R_d$ and calculate the Equation in the following recursive way

$$\begin{cases} \bar{R}_d^{(i)} = 2\bar{L}\bar{R}_d^{(i-1)} - \bar{R}_d^{(i-2)} \\ \bar{R}_d^{(0)} = R_d \\ \bar{R}_d^{(1)} = \bar{L}R_d \end{cases}$$

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
	node2vec	**36.3**	39.7	41.1	42.0	42.1
	GraRep	34.0	32.5	33.3	33.7	34.1

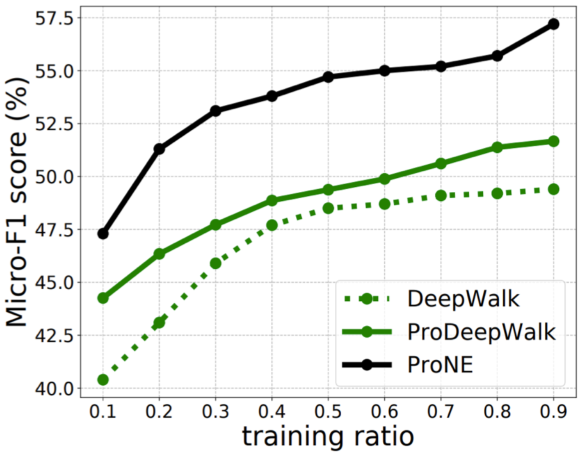
Dataset	training ratio	0.01	0.03	0.05	0.07	0.09
DBLP	DeepWalk	49.3	55.0	57.1	57.9	58.4
	LINE	48.7	52.6	53.5	54.1	54.5
	node2vec	48.9	55.1	57.0	58.0	58.4
	GraRep	50.5	52.6	53.2	53.5	53.8
	HOPE	**52.2**	55.0	55.9	56.3	56.6
	ProNE (SMF)	50.8	54.9	56.1	56.7	57.0
	ProNE	48.8	**56.2**	**58.0**	**58.8**	**59.2**
	($\pm\sigma$)	($\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/ only sparse matrix factorization

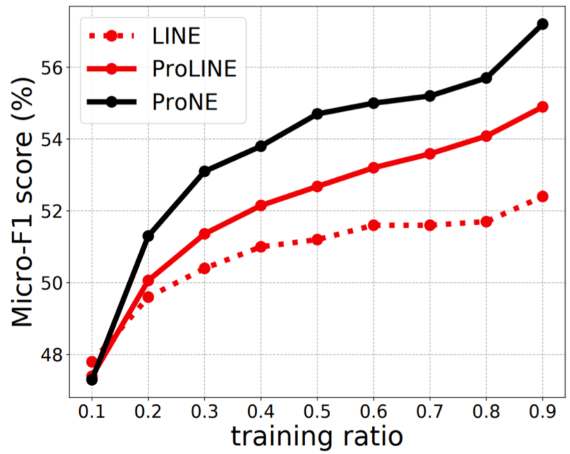
Embed 100,000,000 nodes by one thread:
29 hours with **performance superiority**

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

Spectral Propagation: *k*-way Cheeger

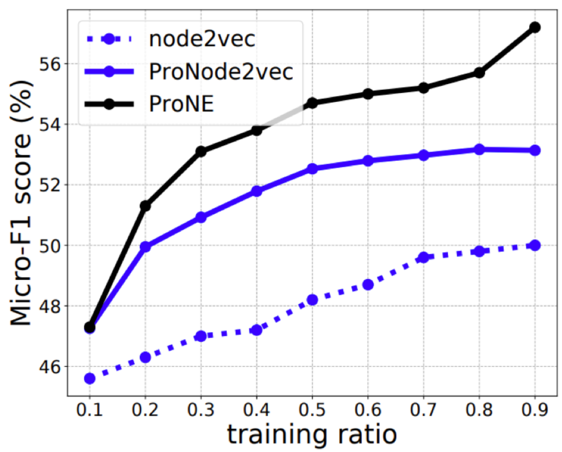


(a) ProDeepWalk

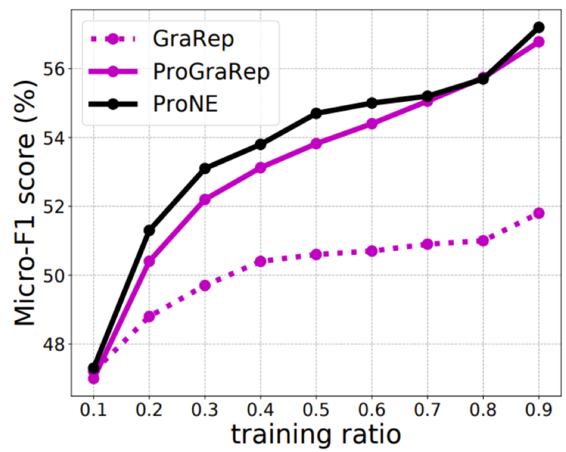


(b) ProLINE

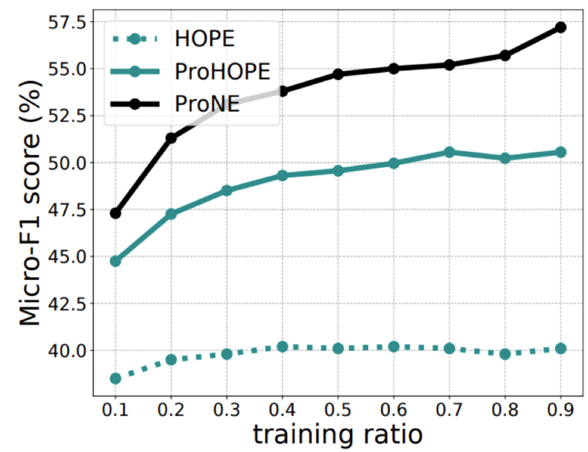
$$\frac{\lambda_k}{2} \leq \rho_G(k) \leq O(k^2)\sqrt{\lambda_k}$$



(c) ProNode2vec

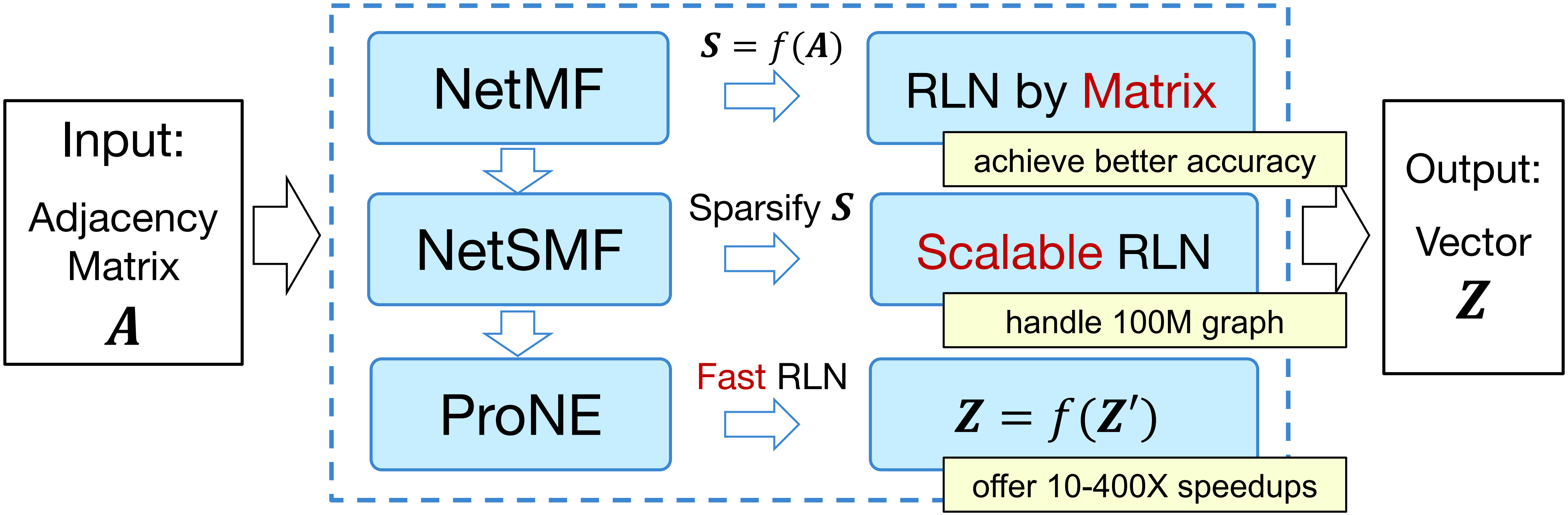


(d) ProGraRep



(e) ProHOPE

Representation Learning on Networks



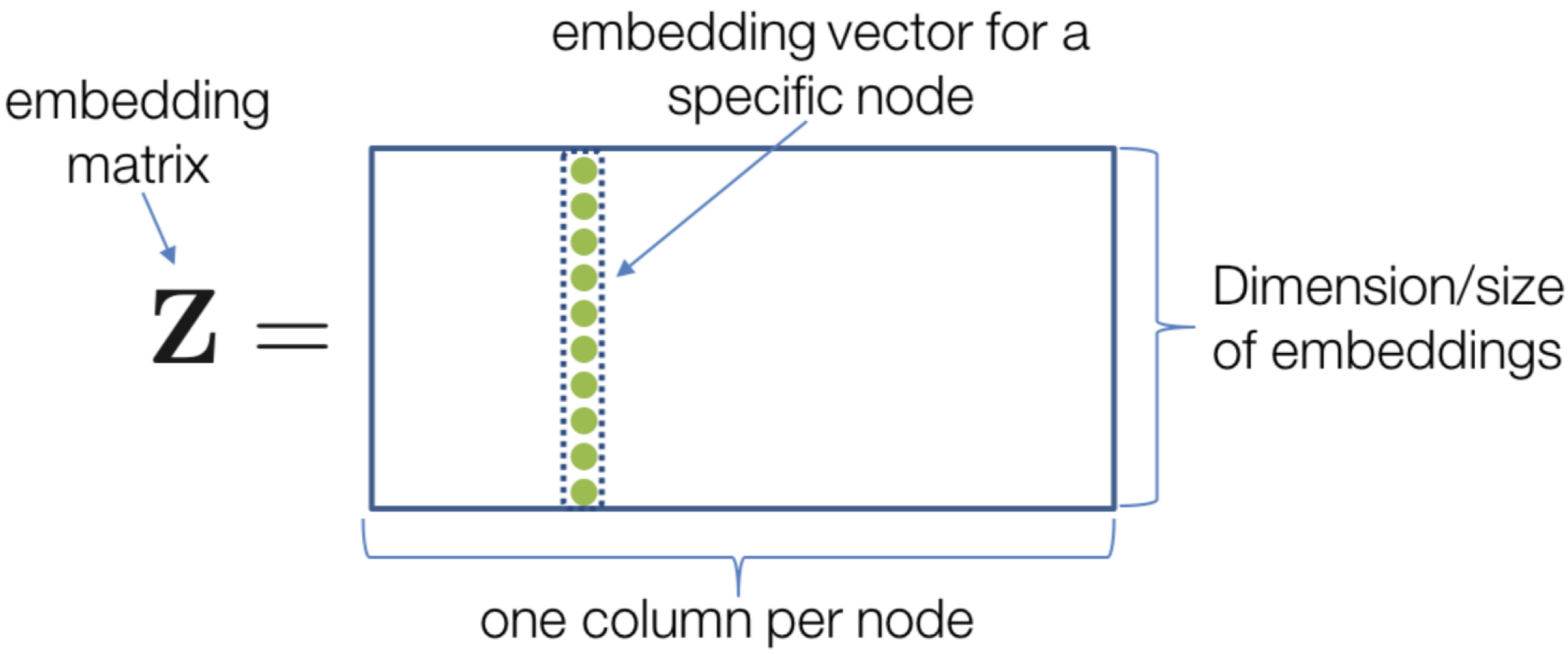
1. Qiu et al. Network embedding as matrix factorization: unifying deepwalk, line, pte, and node2vec. *WSDM'18.* **The most cited paper in WSDM'18 as of May 2019**
2. J. Qiu, Y. Dong, H. Ma, J. Li, C. Wang, K. Wang, and J. Tang. NetSMF: Large-Scale Network Embedding as Sparse Matrix Factorization. WWW'19.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

Agenda

- Network Embedding
- Revisiting Network Embedding
- Graph Neural Network
- Revisiting Graph Neural Network
- GNN&Reasoning
- Revisiting GNN&Reasoning

From Shallow to Deep

- So far we have focused on shallow encoders, i.e. embedding lookups:

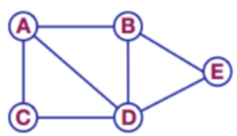


From Shallow to Deep

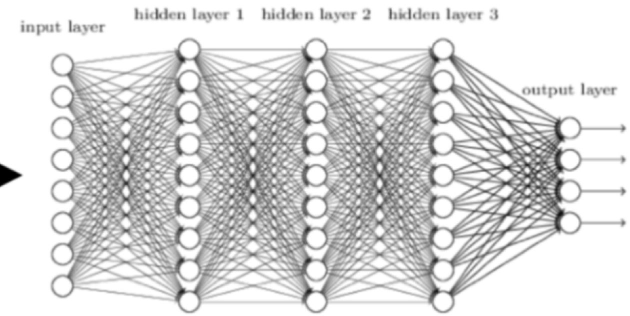
- Limitations of shallow encoding:
  - O(|V|) parameters are needed: there no parameter sharing and every node has its own unique embedding vector
  - Inherently "transductive": It is impossible to generate embeddings for nodes that were not seen during training
  - Do not incorporate node features: Many graphs have features that we can and should leverage.
- We will now discuss deeper methods based on graph neural networks, i.e., encoder is complex function that depends on graph structure.

A Naive Approach With Deep Neural Network

- Taking adjacency matrix A and feature matrix X into deep(fully connected neural network).



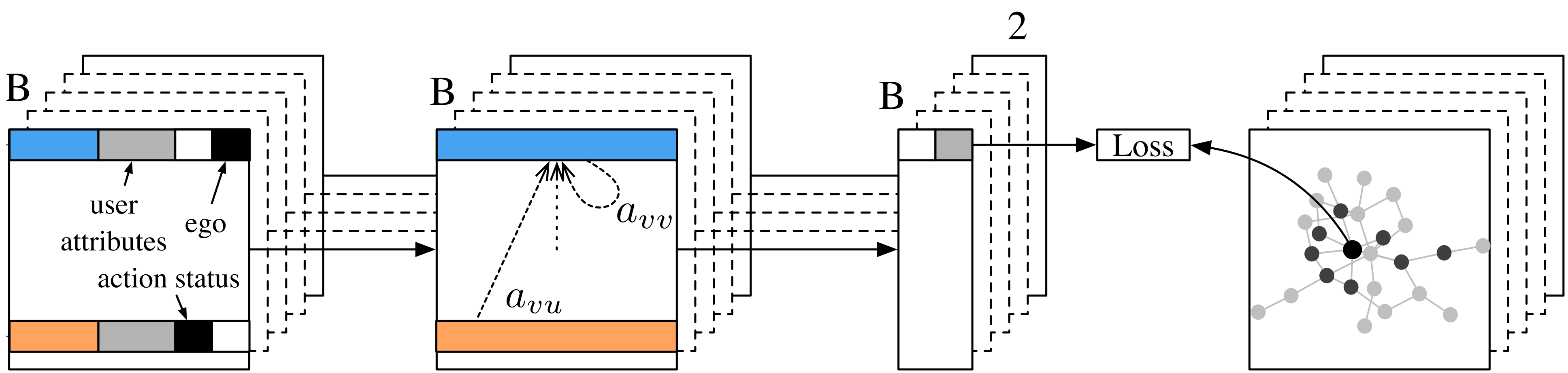
$$\begin{array}{c} \\ A \\ B \\ C \\ D \\ E \end{array} \begin{array}{c} \begin{array}{ccccccc} A & B & C & D & E & \multicolumn{2}{c}{\text{Feat}} \end{array} \\ \begin{pmatrix} 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \end{pmatrix} \end{array}$$



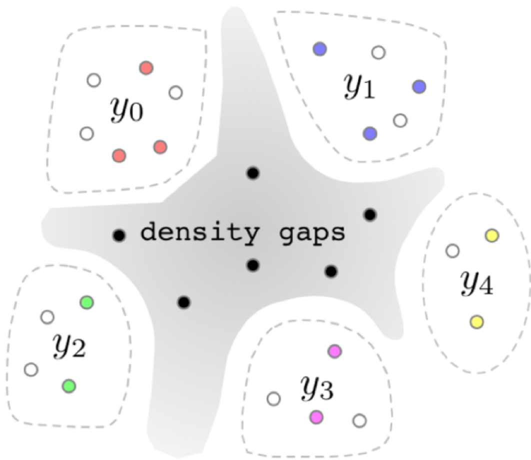
- But there are a huge number of parameters and you have to re-train the model if the network structure changes.
- So we need weight sharing(CNNs)!

GCN

- GCNs can be considered as a simplification of the traditional graph spectral methods
- The common strategy is to model a node's neighborhood as the receptive field and then apply the *graph convolution* operation



Recent GCN Research



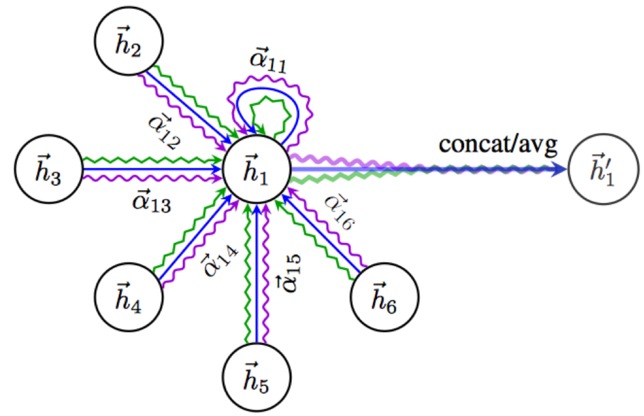
2018 GraphSGAN

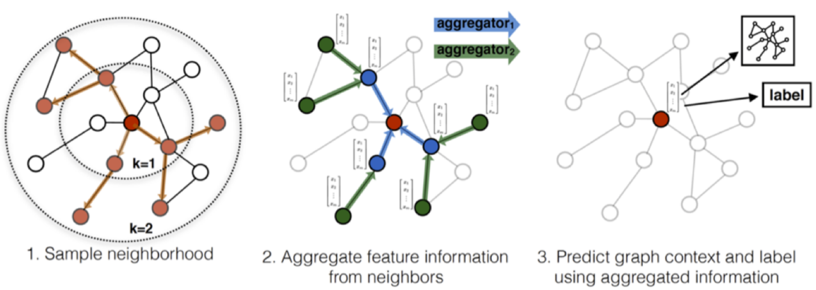
FastGCN 2018

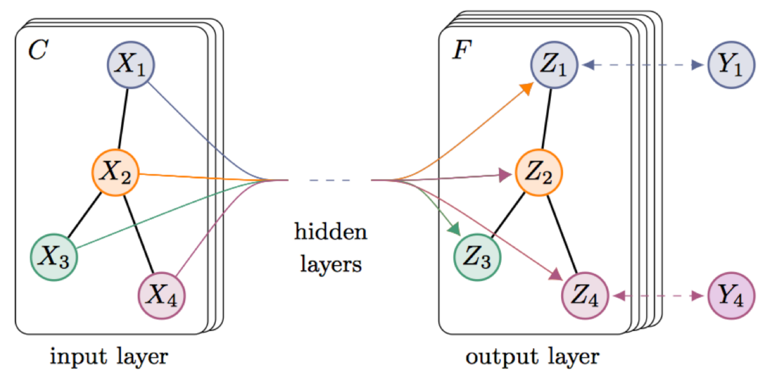
2018 GAT

GraphSAGE 2017

2017 GCN

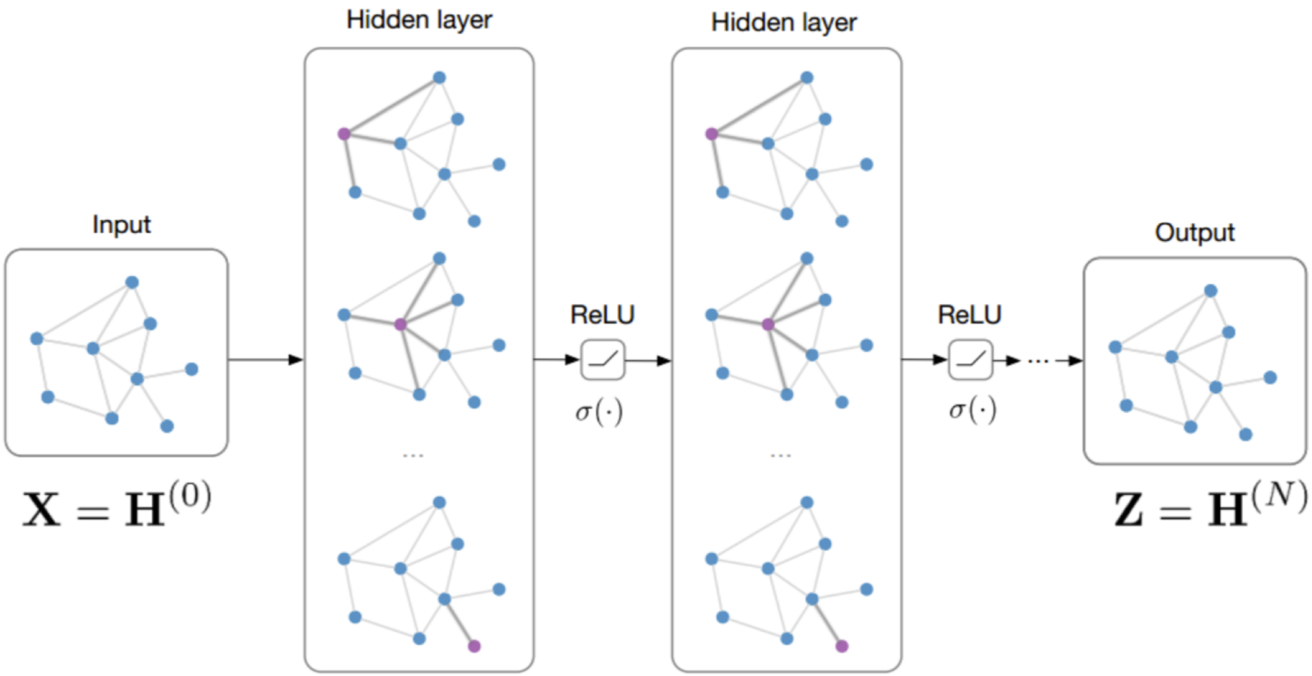






GNN/GCN Model Architecture

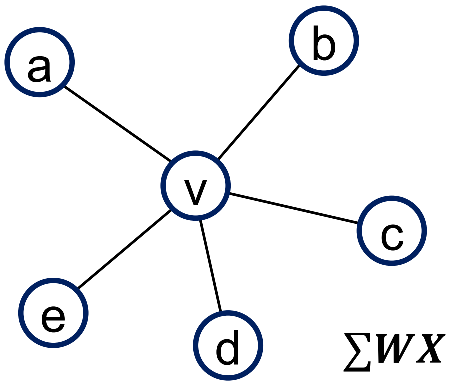
- **Input**: preprocess adjacency matrix $\hat{\mathbf{A}}$ and feature matrix $\mathbf{X} \in R^{N \times E}$



$$\mathbf{H}^{(l+1)} = \sigma\left(\hat{\mathbf{A}}\mathbf{H}^{(l)}\mathbf{W}^{(l)}\right)$$

- where $\hat{\mathbf{A}} = \tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}$, $\tilde{A} = A + I_N$ is the adjacency matrix of graph G with added self-connections and $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$

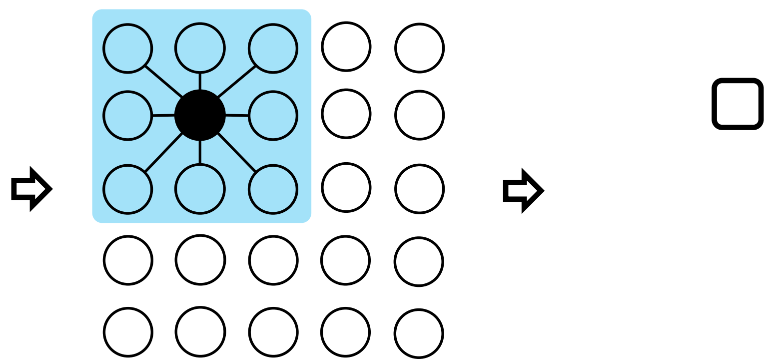
The Core of Graph Neural Networks



$$\boldsymbol{h}_v = f(\boldsymbol{h}_a, \boldsymbol{h}_b, \boldsymbol{h}_c, \boldsymbol{h}_d, \boldsymbol{h}_e)$$

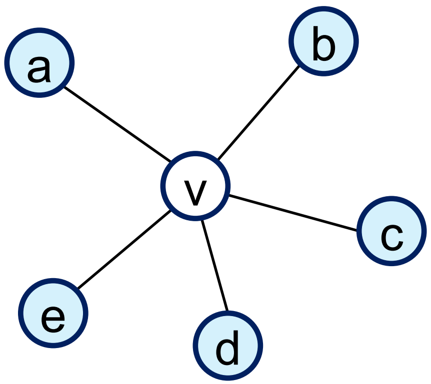
- **Neighborhood Aggregation:**
  - Aggregate neighbor information and pass into a neural network
  - It can be viewed as a center-surround filter in CNN---graph convolutions!

Convolutional neural network



$\sum WX$

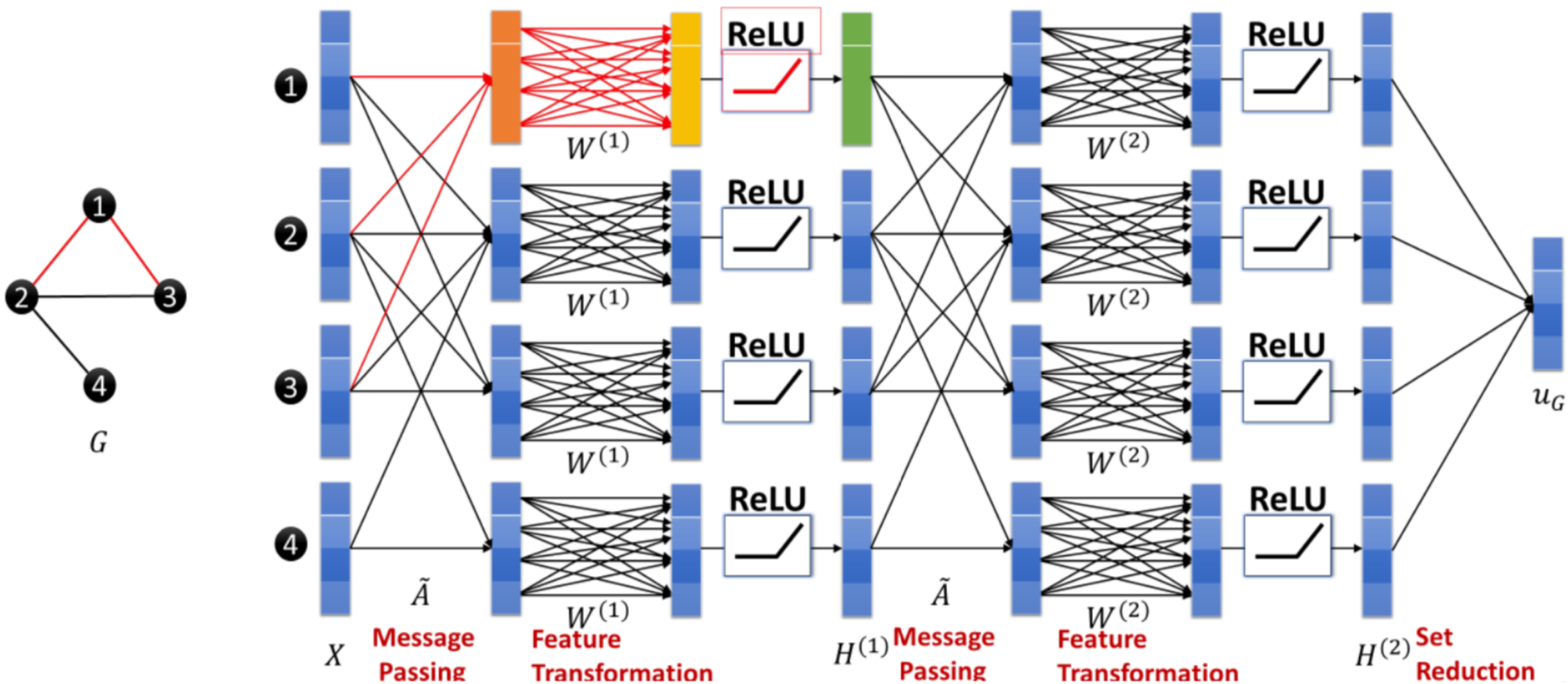
GNN: Graph Convolutional Networks



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

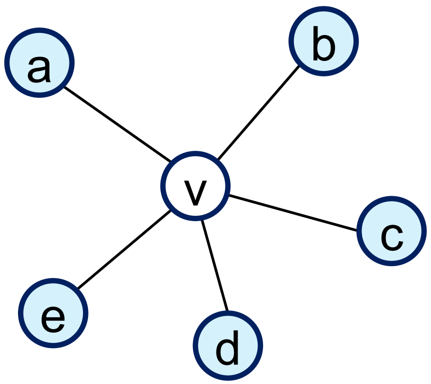
1. Kipf et al. Semi-supervised Classification with Graph Convolutional Networks. ICLR 2017

A Toy Example



the example is from Yizhou's slides

GNN: Graph Convolutional Networks



parameters in layer k

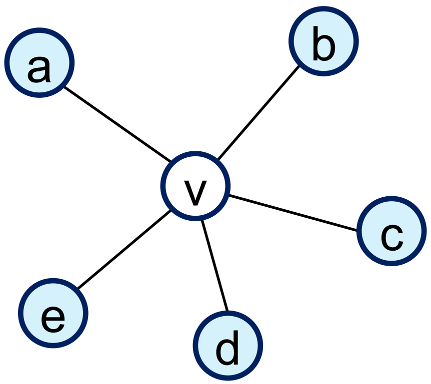
Non-linear activation function (e.g., ReLU)

$$h_v^k = \sigma\left(W_k \sum_{u \in N(v) \cup v} \frac{h_u^{k-1}}{\sqrt{|N(u)||N(v)|}}\right)$$

node v's embedding at layer k

the neighbors of node v

GNN: Graph Convolutional Networks



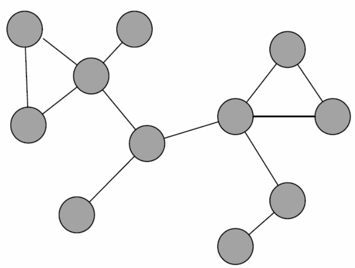
$$\boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{A}\boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{H}^{(k)}\boldsymbol{W}^{(k)}$$

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

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

1. Kipf et al. Semi-supervised Classification with Graph Convolutional Networks. ICLR 2017

GNN: Graph Convolutional Networks



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

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

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

Input

Output

GNN: Graph Convolutional Networks

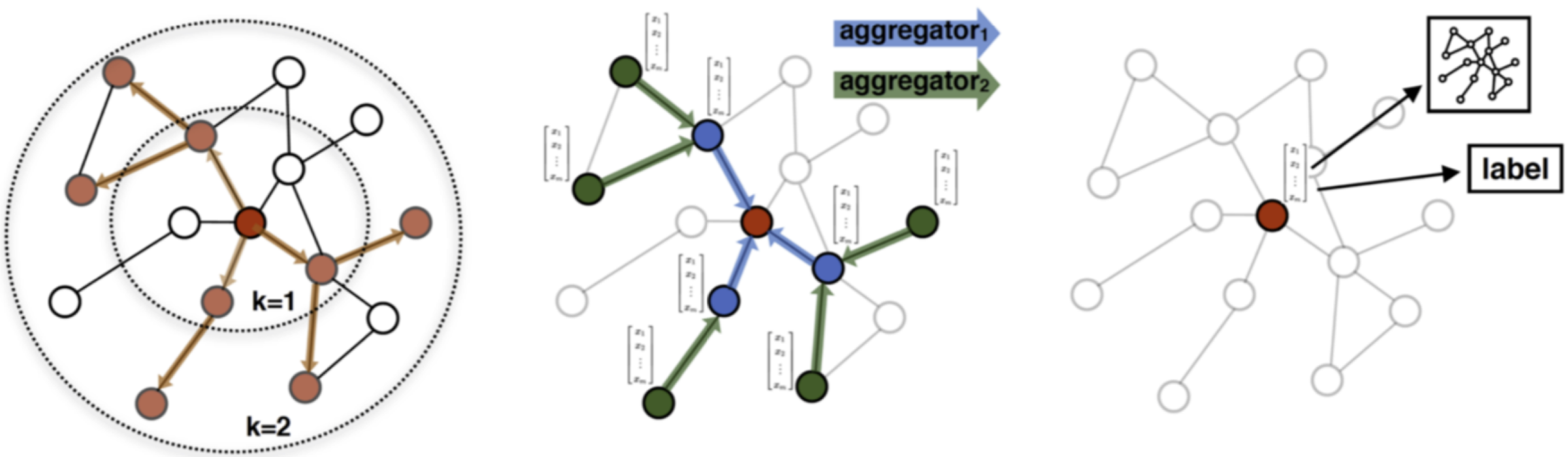
$$\boldsymbol{H}^{k} = \sigma\left(\boldsymbol{D}^{-\frac{1}{2}}(\boldsymbol{A}+\boldsymbol{I})\boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{H}^{(k)}\boldsymbol{W}^{(k)}\right)$$

GCN is one way of neighbor aggregations

- **GraphSage**
- Graph Attention
- … …

GraphSAGE Model Arthitecture

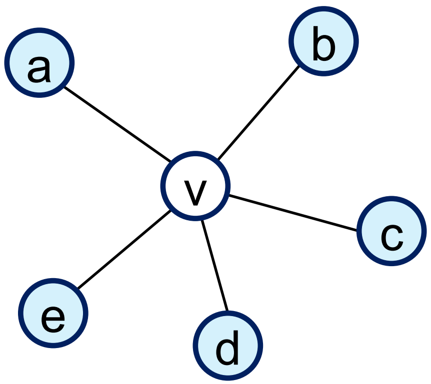
- Idea: Node's neighborhood defines a computation graph.
- Learn how to propagate information across the graph to compute node features



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

1. Hamilton et al. Inductive Representation Learning on Large Graphs. NIPS 2017

GraphSage



GCN

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

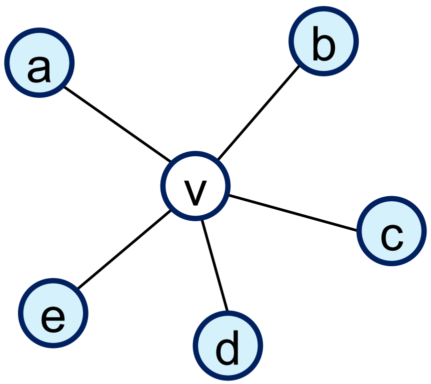
GraphSage

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

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. Slide snipping from “Hamiltion & Tang, AAAI 2019 Tutorial on Graph Representation Learning”

GraphSage



GCN

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

GraphSage

Instead of summation, it concatenate neighbor & self embeddings

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

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

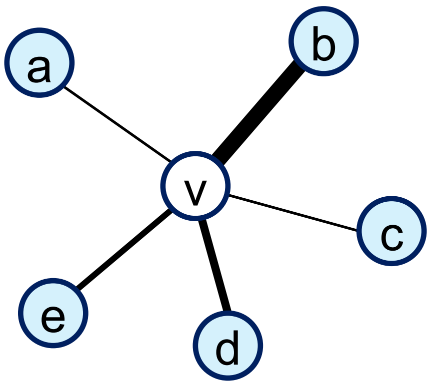
1. Hamilton et al. Inductive Representation Learning on Large Graphs. NIPS 2017
2. Slide snipping from "Hamiltion & Tang, AAAI 2019 Tutorial on Graph Representation Learning"

Performance

Table 1: Prediction results for the three datasets (micro-averaged F1 scores). Results for unsupervised and fully supervised GraphSAGE are shown. Analogous trends hold for macro-averaged scores.

	Citation		Reddit		PPI	
Name	Unsup. F1	Sup. F1	Unsup. F1	Sup. F1	Unsup. F1	Sup. F1
Random	0.206	0.206	0.043	0.042	0.396	0.396
Raw features	0.575	0.575	0.585	0.585	0.422	0.422
DeepWalk	0.565	0.565	0.324	0.324	—	—
DeepWalk + features	0.701	0.701	0.691	0.691	—	—
GraphSAGE-GCN	0.742	0.772	**0.908**	0.930	0.465	0.500
GraphSAGE-mean	0.778	0.820	0.897	0.950	0.486	0.598
GraphSAGE-LSTM	0.788	0.832	**0.907**	**0.954**	0.482	**0.612**
GraphSAGE-pool	**0.798**	**0.839**	0.892	0.948	**0.502**	0.600
% gain over feat.	39%	46%	55%	63%	19%	45%

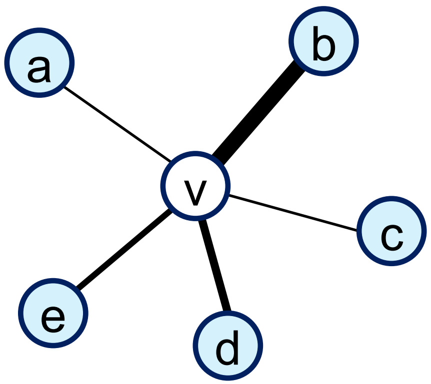
Graph Attention Networks



Realistically, neighbors play different influences

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

Graph Attention Networks



GCN

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

Graph Attention

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

Learned attention weights

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

GAT Results

Transductive

Method	Cora	Citeseer	Pubmed
MLP	55.1%	46.5%	71.4%
ManiReg (Belkin et al., 2006)	59.5%	60.1%	70.7%
SemiEmb (Weston et al., 2012)	59.0%	59.6%	71.7%
LP (Zhu et al., 2003)	68.0%	45.3%	63.0%
DeepWalk (Perozzi et al., 2014)	67.2%	43.2%	65.3%
ICA (Lu & Getoor, 2003)	75.1%	69.1%	73.9%
Planetoid (Yang et al., 2016)	75.7%	64.7%	77.2%
Chebyshev (Defferrard et al., 2016)	81.2%	69.8%	74.4%
GCN (Kipf & Welling, 2017)	81.5%	70.3%	**79.0%**
MoNet (Monti et al., 2016)	81.7 ± 0.5%	—	78.8 ± 0.3%
GCN-64*	81.4 ± 0.5%	70.9 ± 0.5%	**79.0** ± 0.3%
GAT (ours)	**83.0** ± 0.7%	**72.5** ± 0.7%	**79.0** ± 0.3%

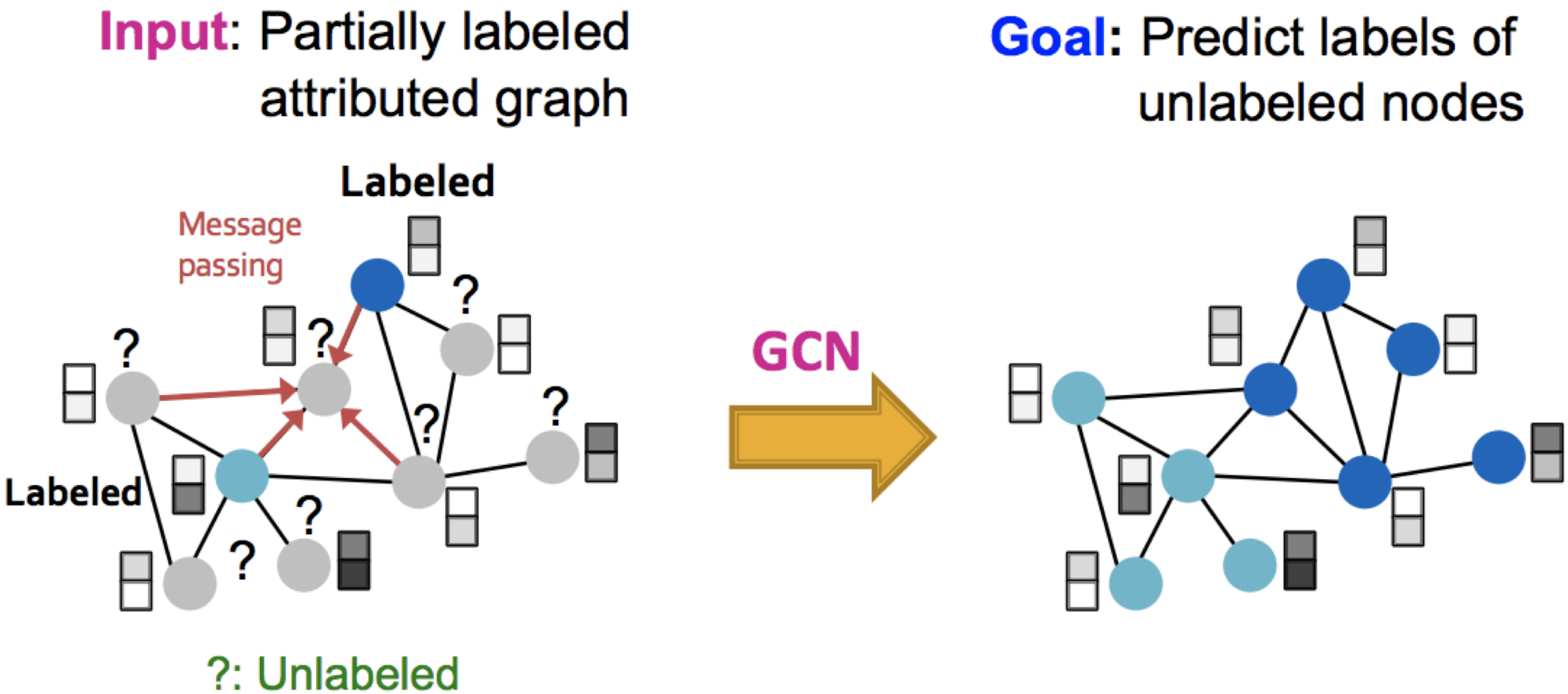
Inductive

Method	PPI
Random	0.396
MLP	0.422
GraphSAGE-GCN (Hamilton et al., 2017)	0.500
GraphSAGE-mean (Hamilton et al., 2017)	0.598
GraphSAGE-LSTM (Hamilton et al., 2017)	0.612
GraphSAGE-pool (Hamilton et al., 2017)	0.600
GraphSAGE*	0.768
Const-GAT (ours)	0.934 ± 0.006
GAT (ours)	**0.973** ± 0.002

Agenda

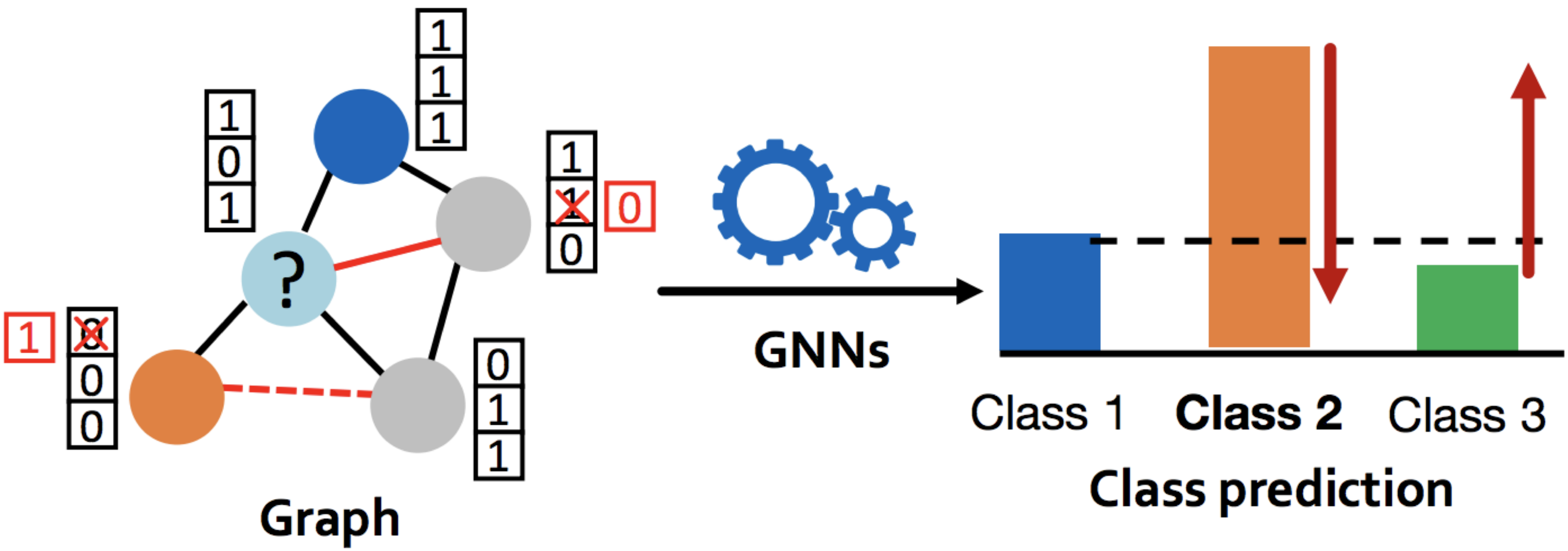
- Network Embedding
- Revisiting Network Embedding
- Graph Neural Network
- Revisiting Graph Neural Network
- GNN&Reasoning
- Revisiting GNN&Reasoning

Setting: Semi-supervised Learning on Graphs



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.

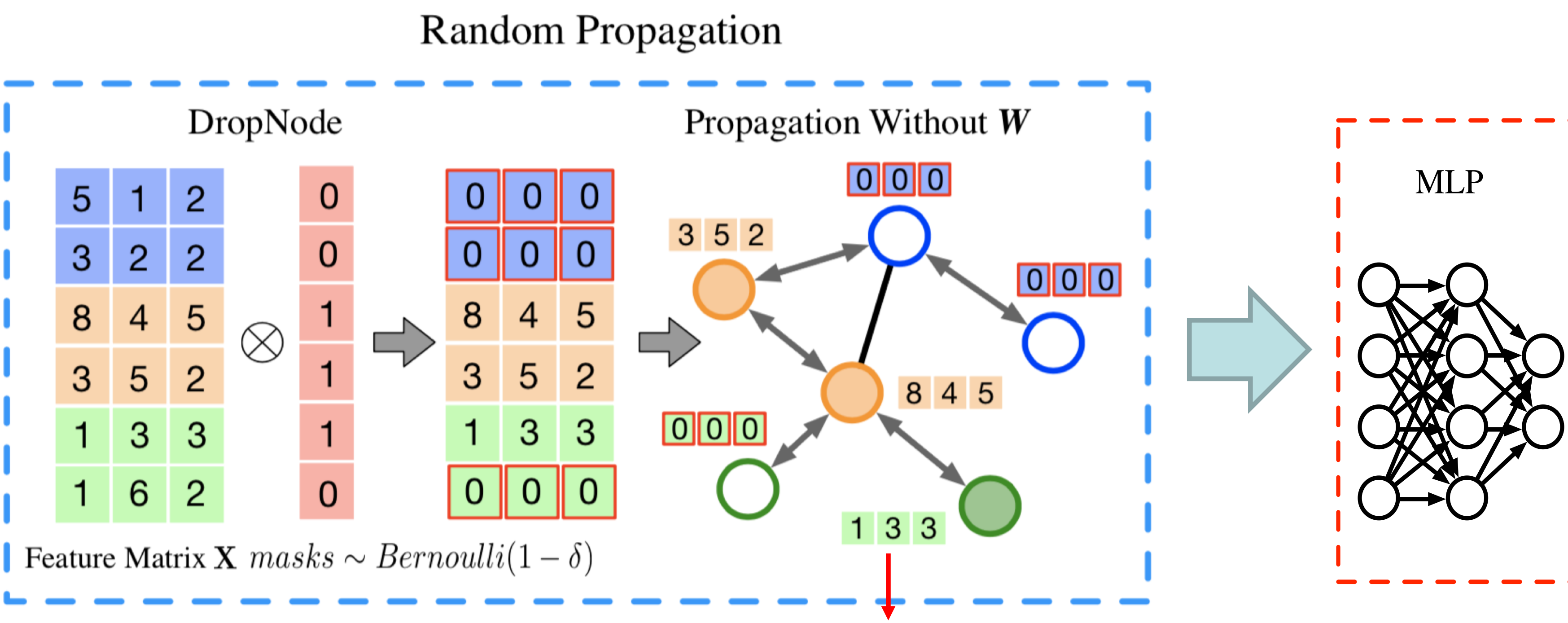


Motivation: some problems of GCN

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

Grand: Graph Random Neural Network

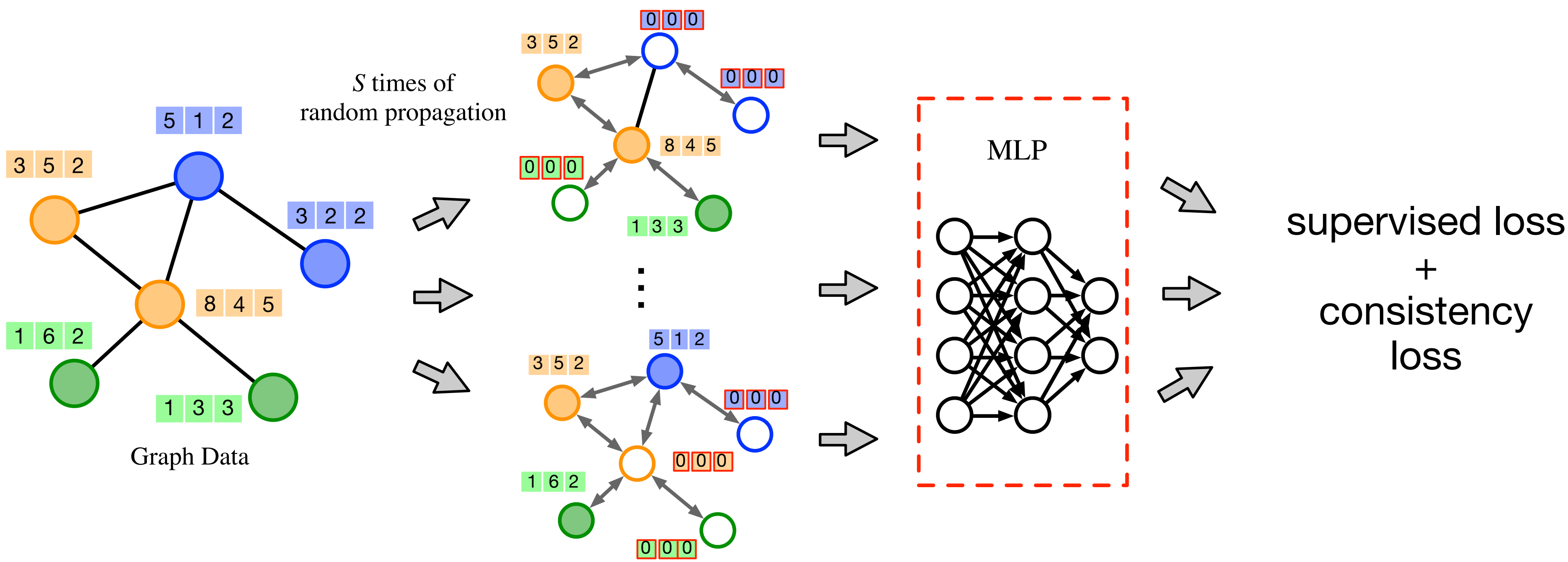
- Random Propagation (DropNode + Propagation):
  - Each node is enabled to be not sensitive to specific neighborhoods.
  - Decouple propagation from feature transformation.



$$\overline{\mathbf{X}}^{(s)} = \frac{1}{K+1}\sum_{k=0}^{K}\hat{\mathbf{A}}^k\widetilde{\mathbf{X}}^{(s)}.$$

Graph Random Neural Network(Grand)

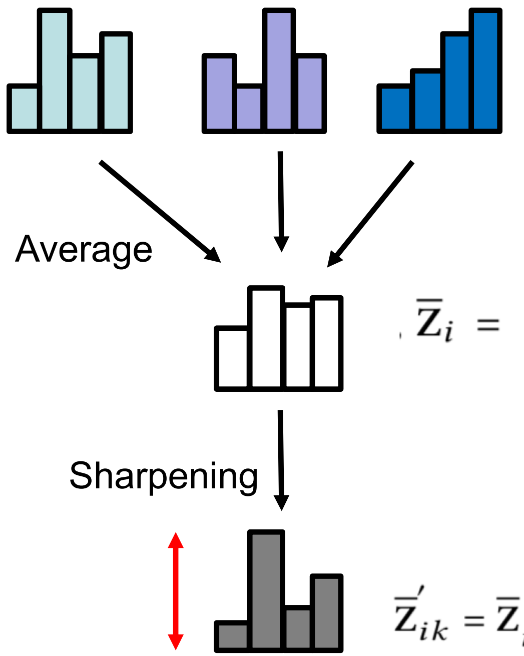
- Consistency Regularized Training:
  - Random propagation serves as graph data augmentation
  - Consistency Regularization: Optimizing the consistency among *S* augmentations of the graph.



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 $\mathbf{Z}$.

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

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

9: **end while**
10: 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

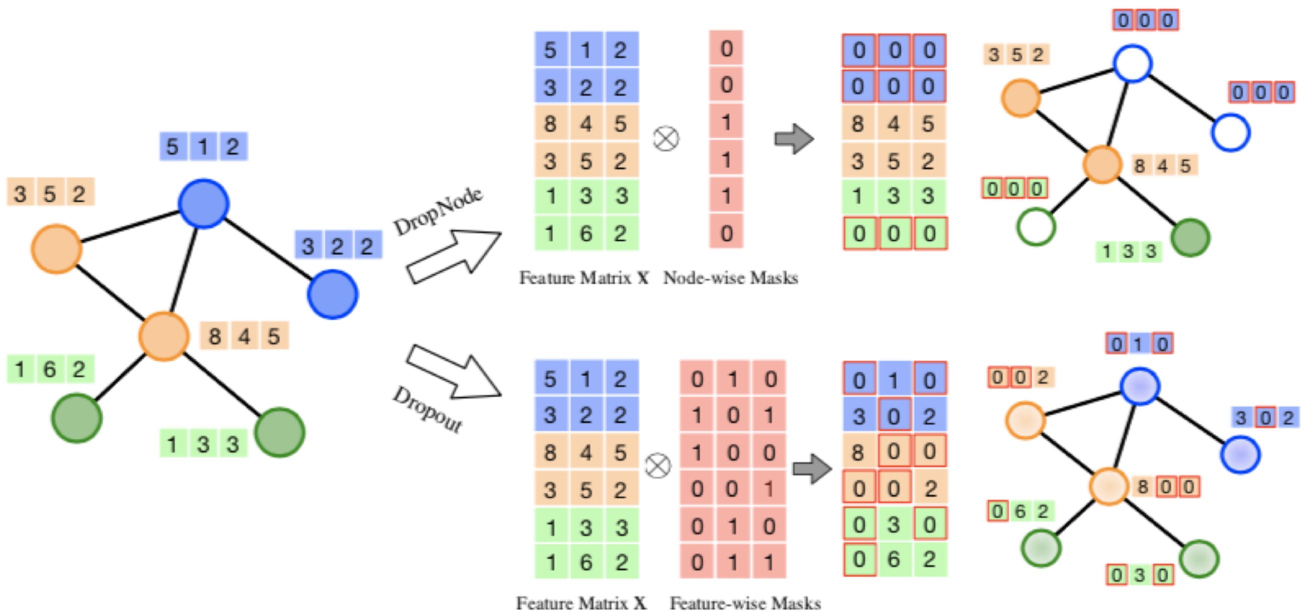


Figure 3: Difference between dropnode and dropout. Dropout drops each element in **X** independently, while DropNode drops the entire features of selected nodes, i.e., the row vectors of **X**, randomly.

- 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

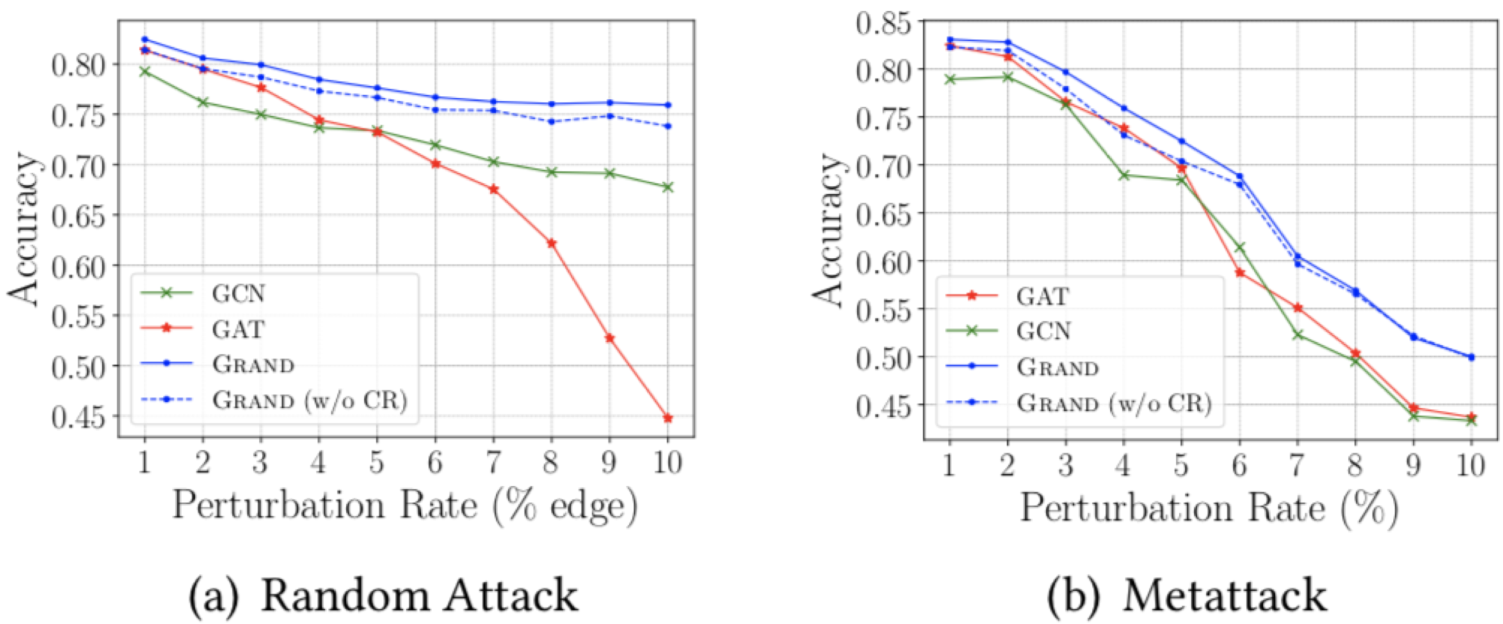
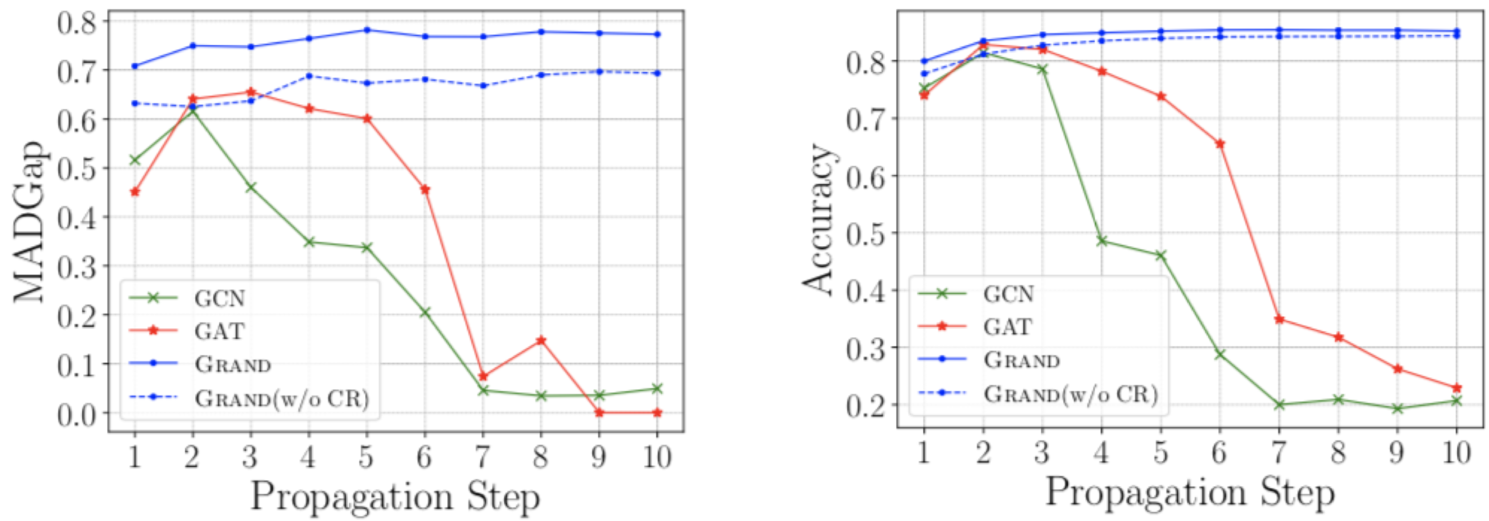


Figure 4: Robustness: results under attacks on Cora.

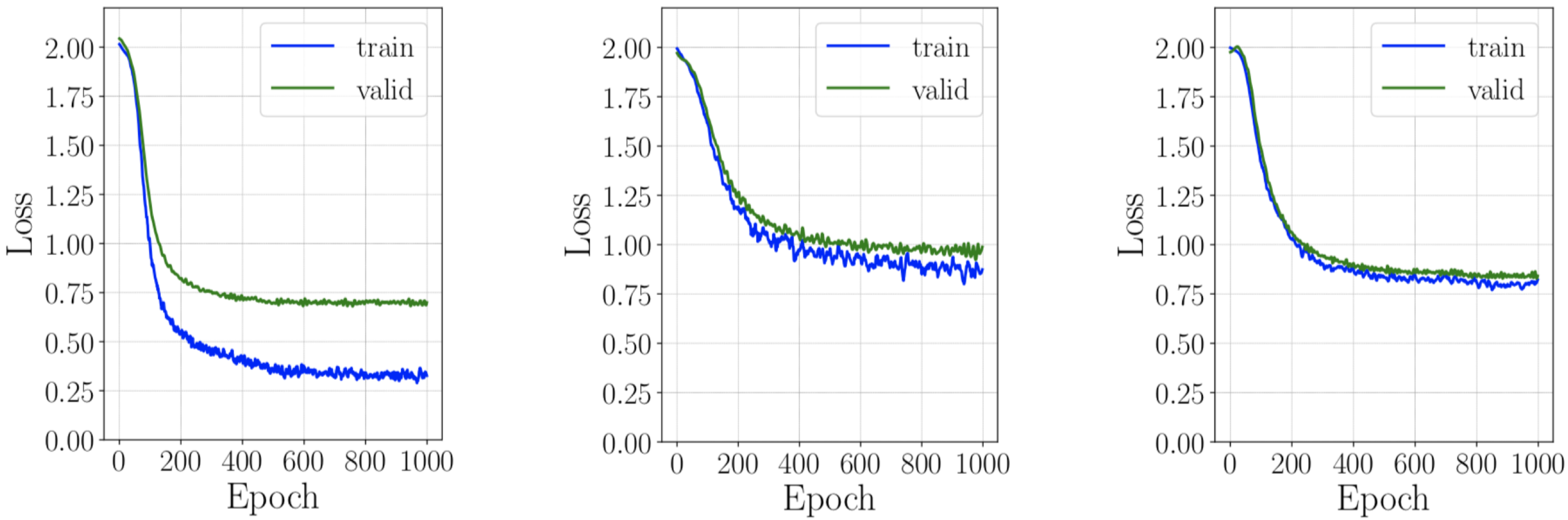
Relieving Over-smoothing



(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

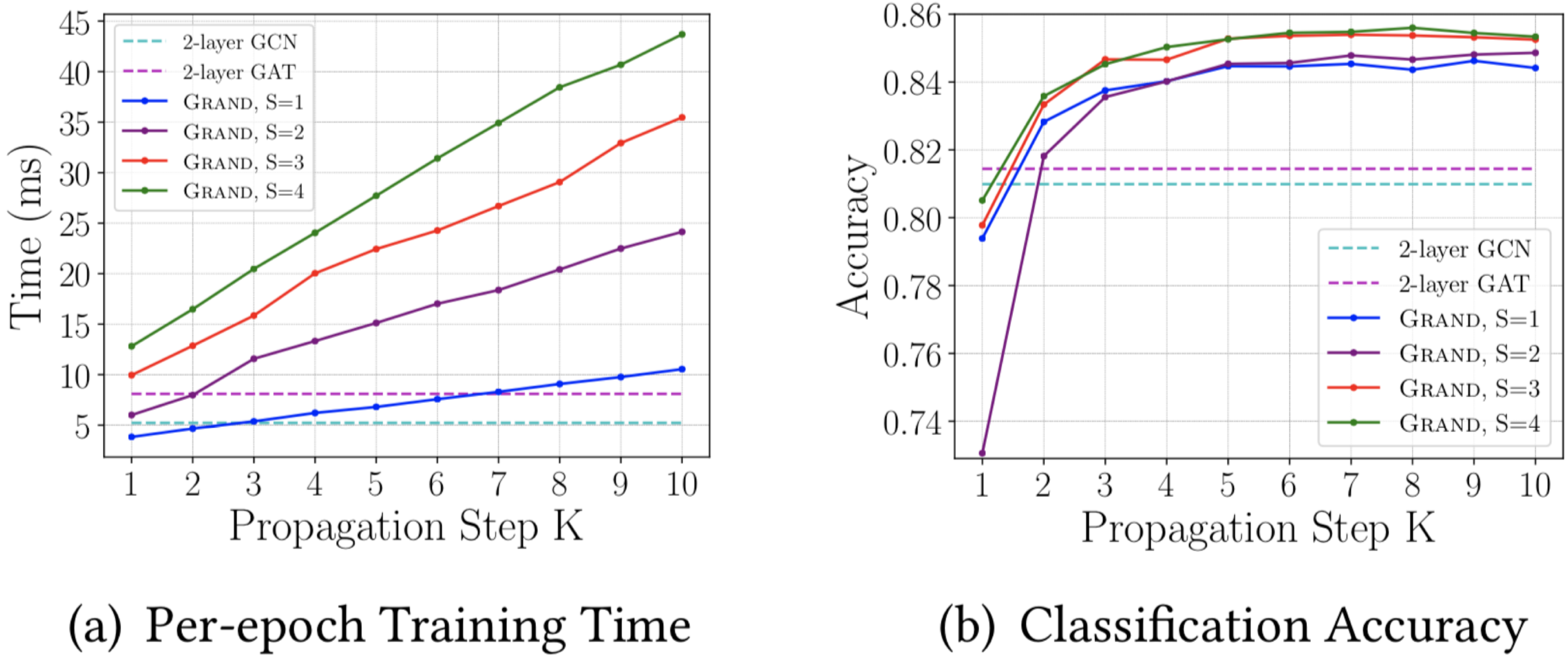


Figure 7: Efficiency Analysis for GRAND.

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

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

convert to matrix form

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

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

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

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

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

Summarization

Methods	Rescaling and Propagation Rules
vanilla GCN	$\mathbf{H}' = \sigma\left(\mathcal{L}\mathbf{H}\mathbf{W}\right)$
GAT	$\mathbf{H}' = \sigma\left(\mathcal{N}\left(\mathbf{P}\mathcal{L} + \mathcal{L}\mathbf{Q}\right)\mathbf{H}\mathbf{W}\right)$
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 $\vec{p}$: $Q_{ii} = \mathrm{sigmoid}(HW\vec{q})_i$
  - 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:

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

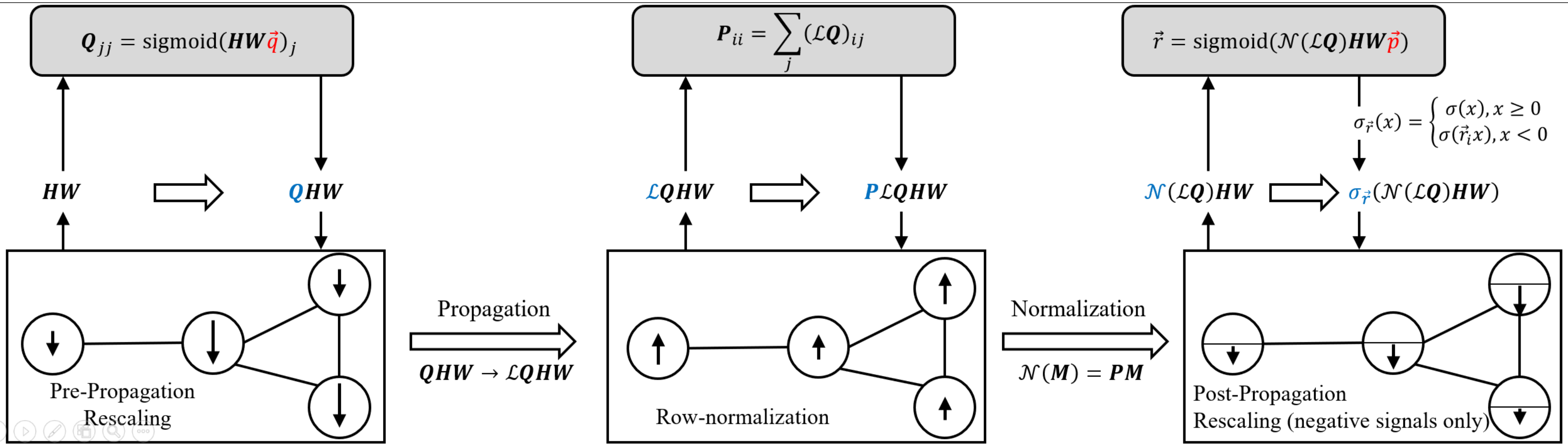
  - Rescaling after propagation $H' = \sigma_{\vec{r}}(\mathcal{L}HW)$
  - $\vec{r}$ are extracted from post-prop feature by a learnable vector $\vec{p}$: $\vec{r}_i = \mathrm{sigmoid}(\mathcal{L}HW\vec{q})_i$
  - Re-normalization are opted out for $\mathrm{xLU}(\cdot)$ activation's output mostly concentrate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

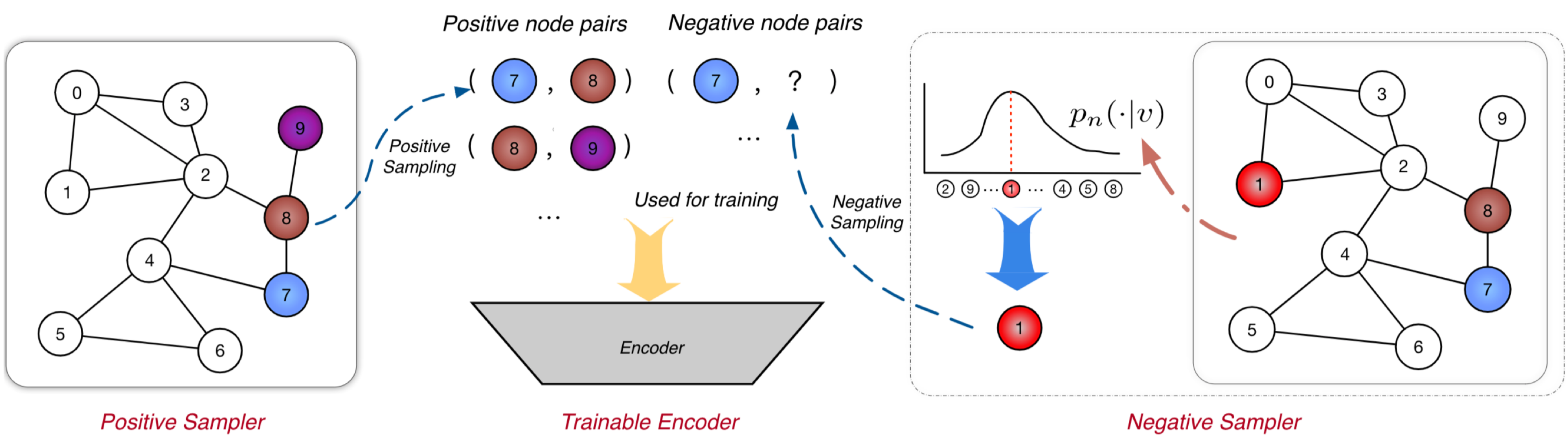
Takeaway Messages & Future Works

- 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:

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

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:

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

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

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

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.

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

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

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

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

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 $p_n(u|v) \propto p_d(u|v)^{\alpha}$,

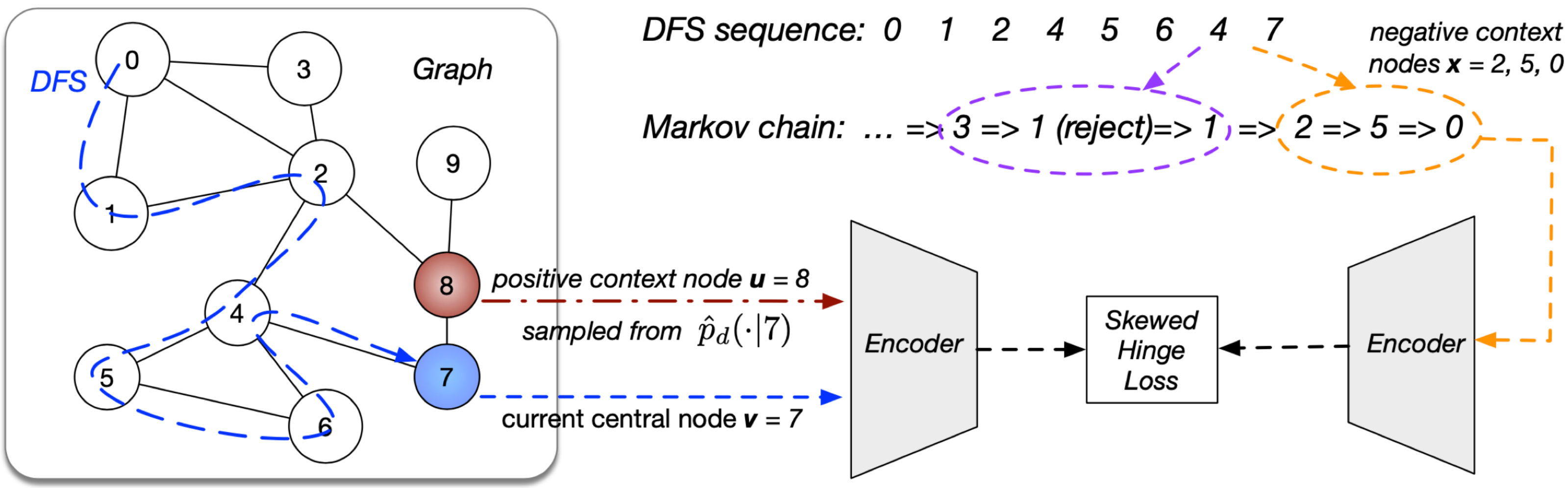
$$\mathbb{E}\left[\|(\theta_T - \theta^*)_u\|^2\right] = \frac{1}{T}\left(\frac{1}{p_d(u|v)}\left(1 + \frac{p_d(u|v)^{1-\alpha}}{c}\right) - 1 - \frac{1}{k}\right)$$

MCNS Model

- self-contrast approximation for positive sampling distribution:
  - replacin 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



$$\mathcal{L} = \max\left(0, E_\theta(v) \cdot E_\theta(x) - E_\theta(u) \cdot E_\theta(v) + \gamma\right)$$

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

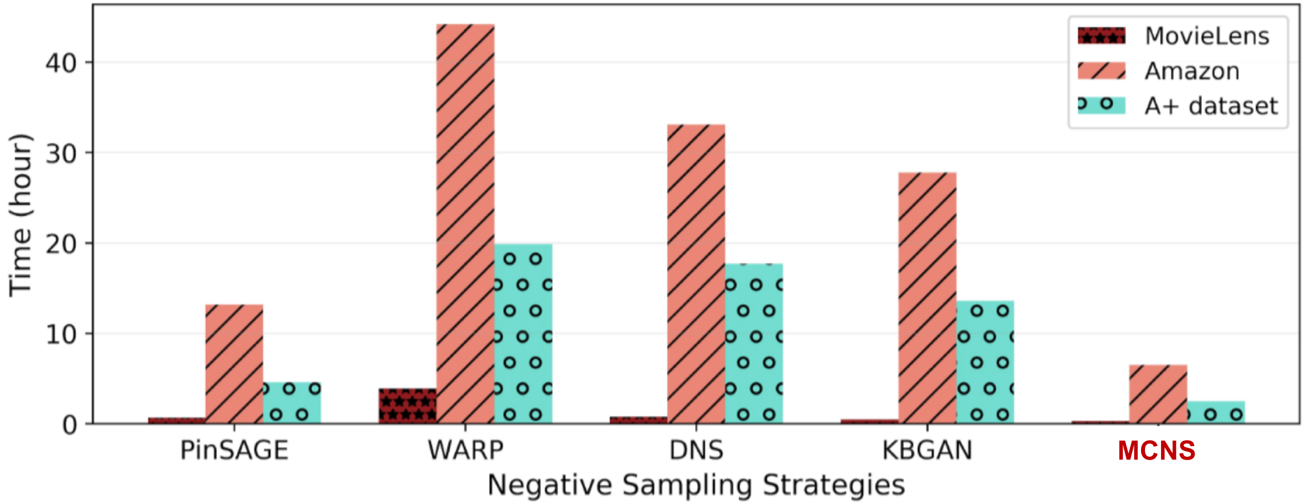
BlogCatalog: 10,312 nodes & 333,983 links

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.

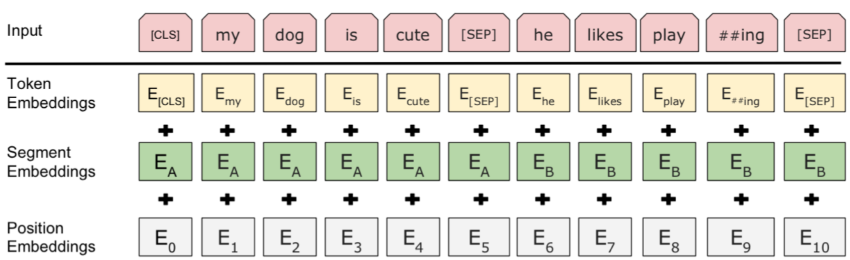
Summary of MCNS

- 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.

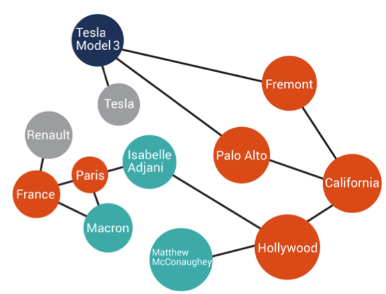
Challenges

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

- Graph Neural Network Pre-Training



+



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

Jiezhong Qiu*, Qibin Chen*, Yuxiao Dong$, Jing Zhang+, Hongxia Yang#, Ming Ding*, Kuansan Wang$, Jie Tang*

*Tsinghua University

$Microsoft Research, Redmond

#DAMO Academy, Alibaba Group

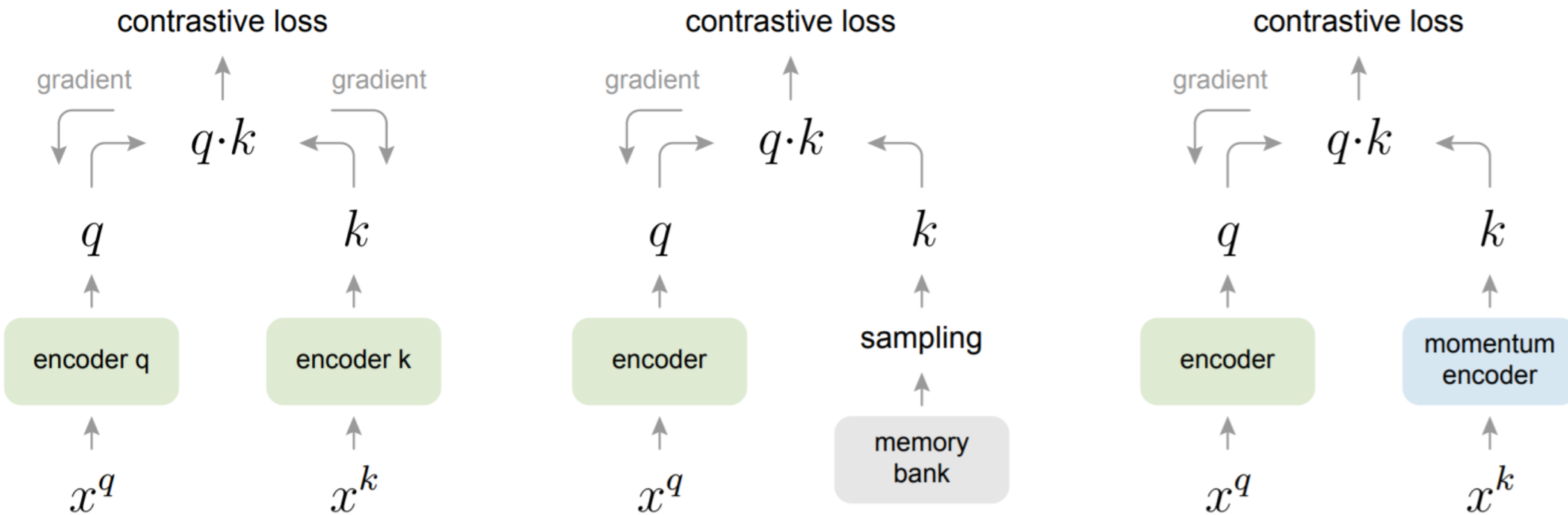
+Renmin University

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 (CVPR 20)

- 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

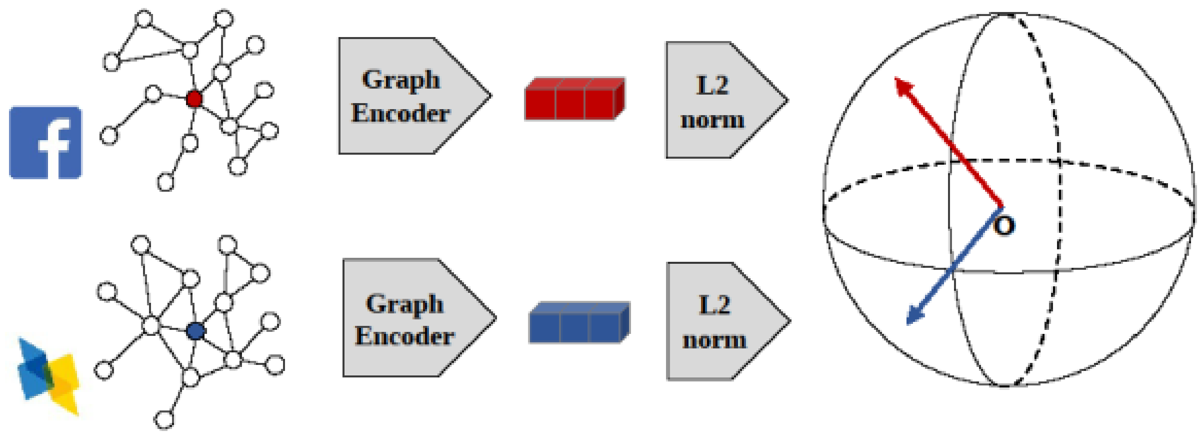
Revisit MoCo



(a) end-to-end (b) memory bank (c) MoCo

Graph Contrastive Coding (GCC)

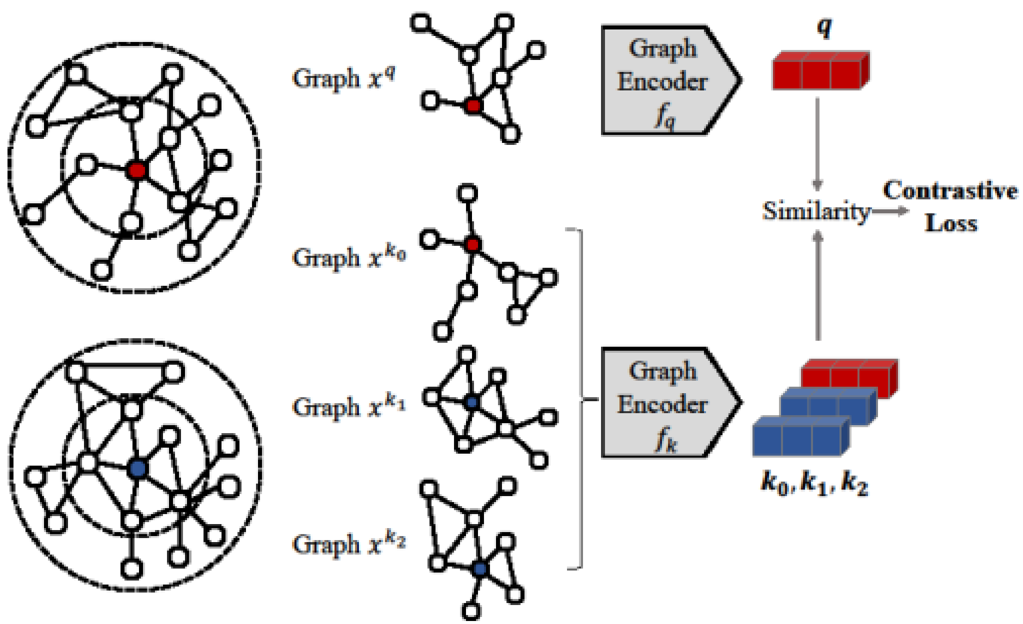
- Problem formulation:
  – Measuring vertex structural similarity



  – 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*.

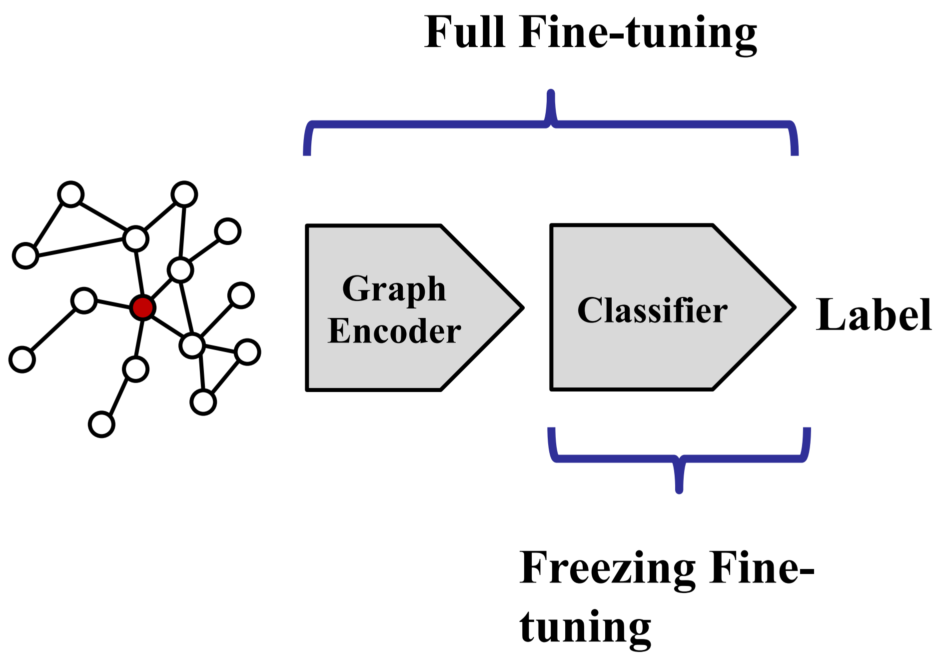
GCC Pre-Training / Fine-tuning

- 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

Agenda

- Network Embedding
- Revisiting Network Embedding
- Graph Neural Network
- Revisiting Graph Neural Network
- GNN&Reasoning
- Revisiting GNN&Reasoning

推理

Question: Who is the director of the 2003 film which has scenes in it filmed at the Quality Cafe in Los Angeles?

Quality Café

The Quality Cafe is a now-defunct diner in Los Angeles, California. The restaurant has appeared as a location featured in a number of Hollywood films, including Old School, Gone in 60 Seconds, ...

Los Angeles

Los Angeles is the most populous city in California, the second most populous city in the United States, after New York City, and the third most populous city in North America.

Alessandro Moschitti

Alessandro Moschitti is a professor of the CS Department of the University of Trento, Italy. He is currently a Principal Research Scientist of the Qatar Computing Research Institute (QCRI)

WIKIPEDIA
The Free Encyclopedia

Old School

Old School is a 2003 American comedy film released by Dream Works Pictures and The Montecito Picture Company and directed by Todd Phillips.

Todd Phillips

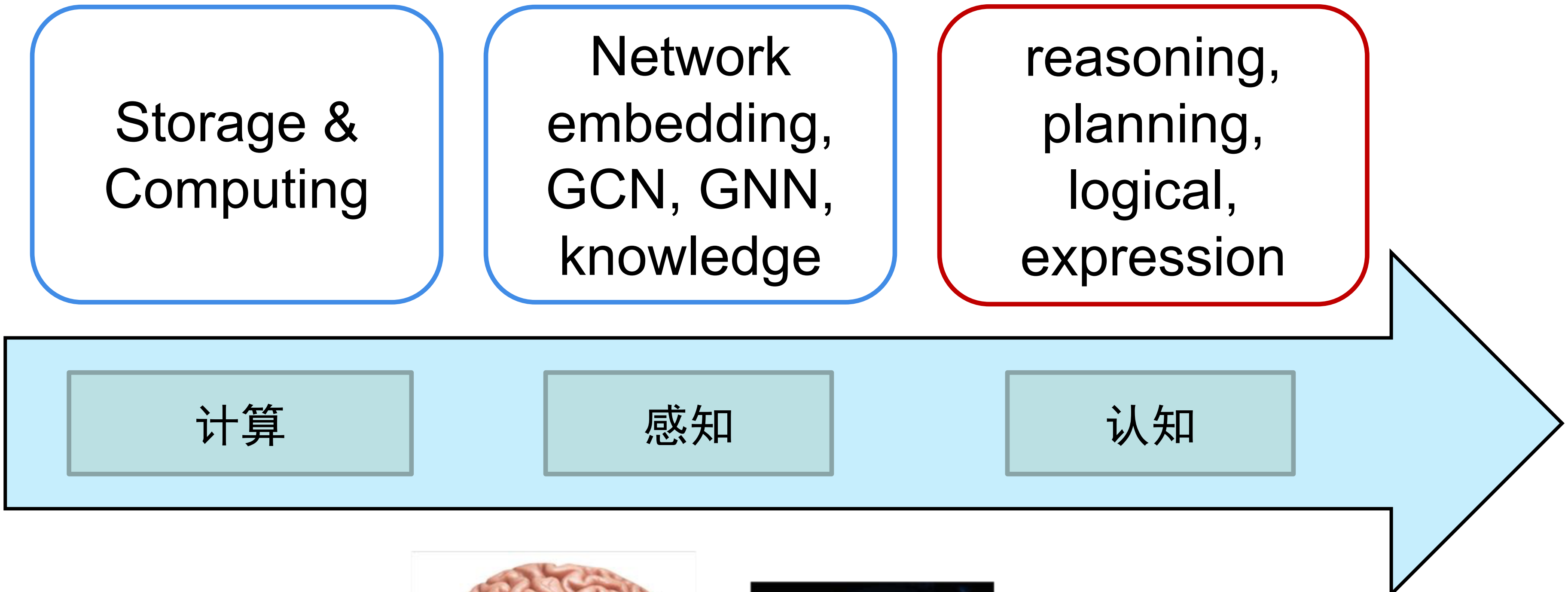
Todd Phillips is an American director, producer, screenwriter, and actor. He is best known for writing and directing films, including Road Trip (2000), Old School (2003), Starsky & Hutch (2004), and The Hangover Trilogy.

Tsinghua University

Tsinghua University is a major research university in Beijing and dedicated to academic excellence and global development. Tsinghua is perennially ranked as one of the top academic institutions in China, Asia, and worldwide...

AI趋势：从感知到认知

- 面向认知的GNN



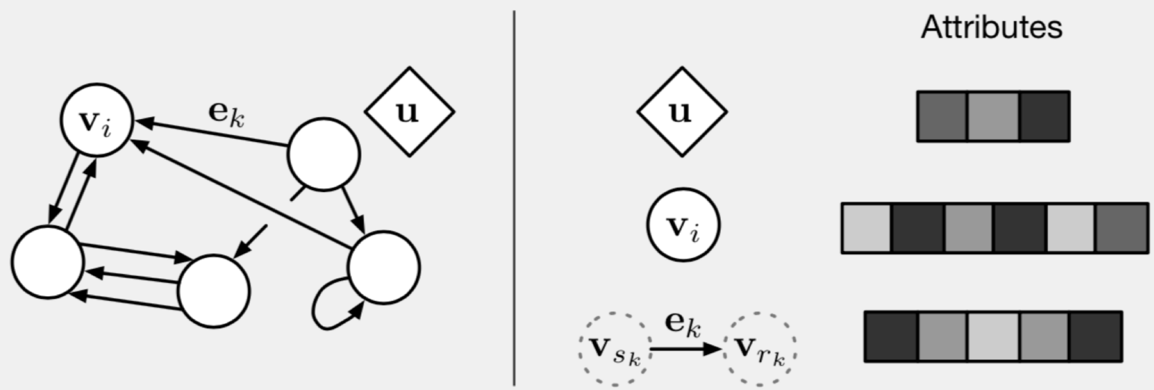
Stochastic vs Deterministic

Uncertainty!

graph_net

- By DeepMind

Box 3: Our definition of "graph"



Here we use "graph" to mean a directed, attributed multi-graph with a global attribute. In our terminology, a node is denoted as $\mathbf{v}_i$, an edge as $\mathbf{e}_k$, and the global attributes as $\mathbf{u}$. We also use s_k and r_k to indicate the indices of the sender and receiver nodes (see below), respectively, for edge k. To be more precise, we define these terms as:

Directed : one-way edges, from a "sender" node to a "receiver" node.
Attribute : properties that can be encoded as a vector, set, or even another graph.
Attributed : edges and vertices have attributes associated with them.
Global attribute : a graph-level attribute.
Multi-graph : there can be more than one edge between vertices, including self-edges.

Figure 2 shows a variety of different types of graphs corresponding to real data that we may be interested in modeling, including physical systems, molecules, images, and text.

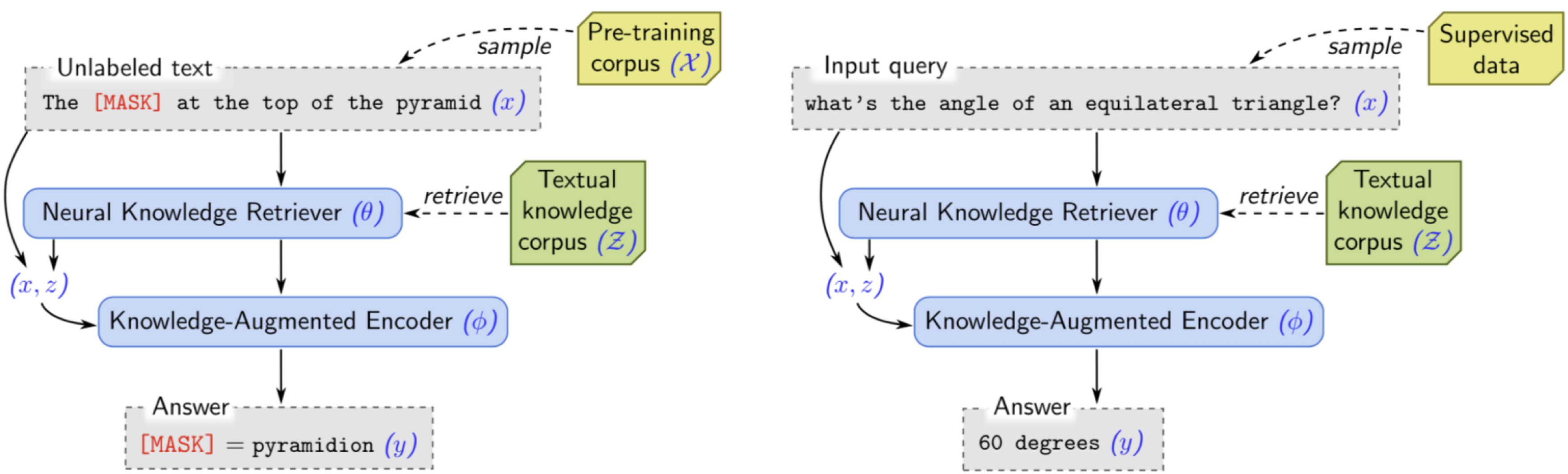
https://arxiv.org/abs/1806.01261

REALM: Retrieval-Augmented LM

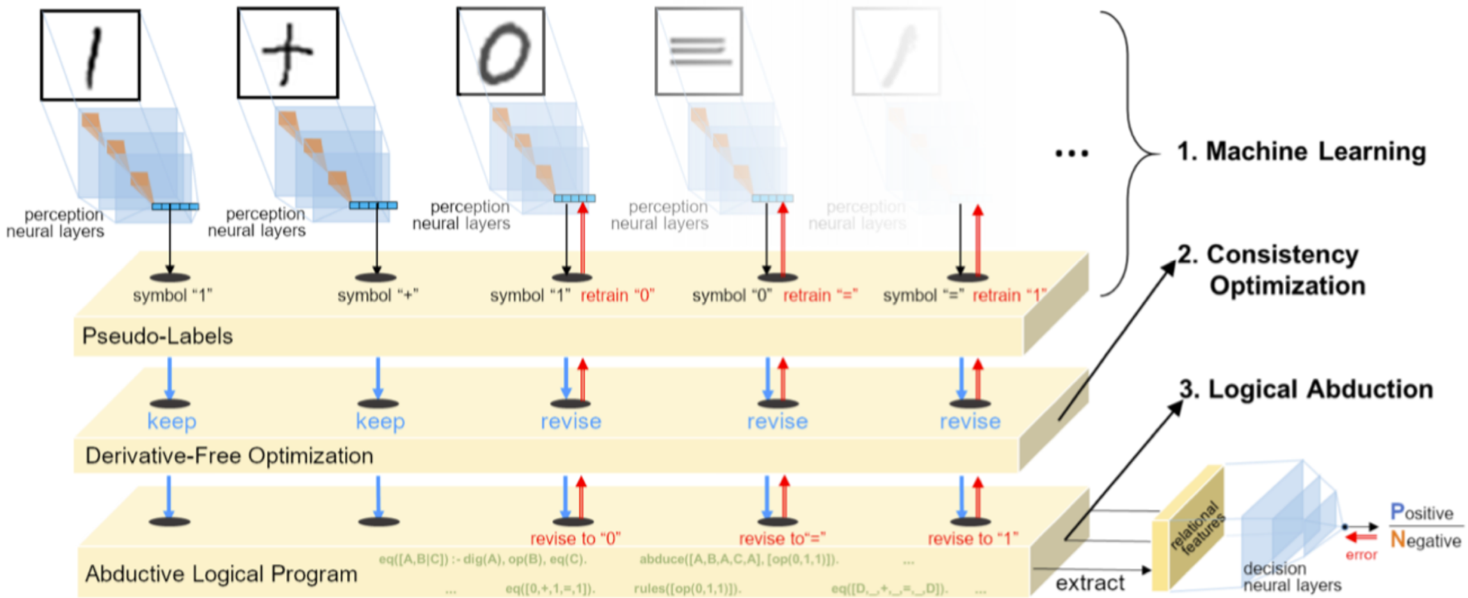
$$\text{BERT}: p(y|x)$$

$$\text{REALM}: p(y|x) = \sum_{z \in \mathcal{Z}} p(y|z,x)p(z|x)$$

- where Z is the supporting set.



Abductive Learning

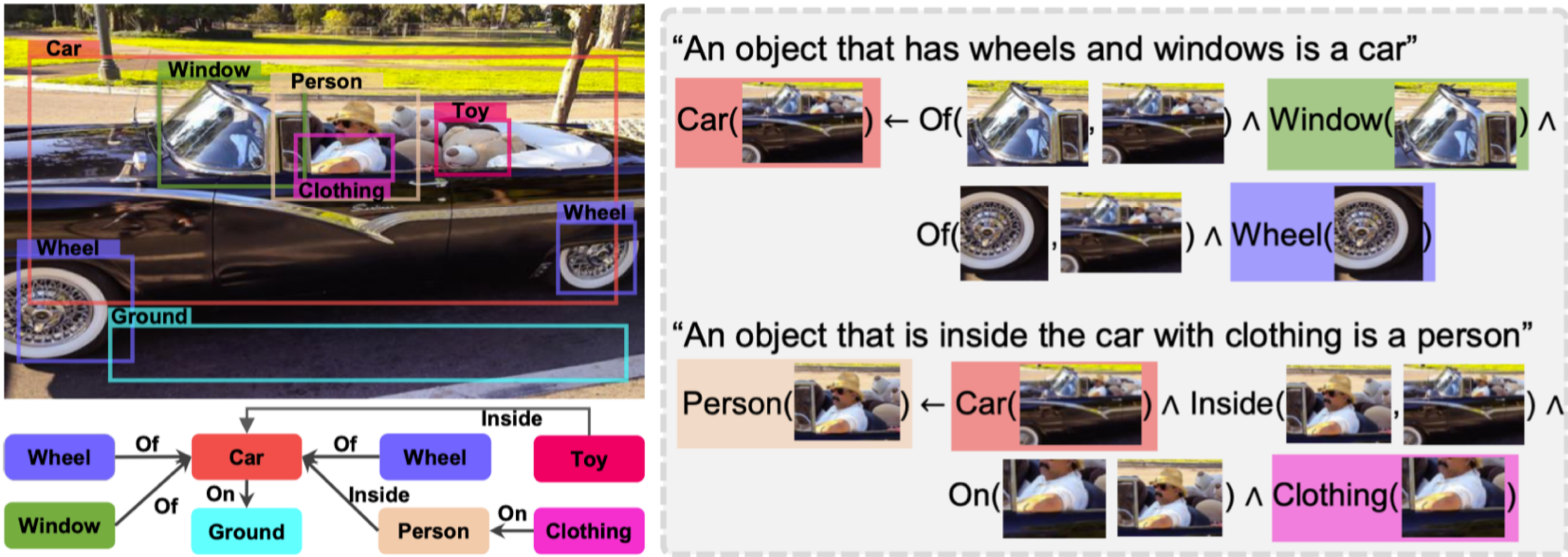


They use deep learning to extract high-level concepts and logical rules to make predictions based on concepts.

The optimization is difficult since it involves non-differentiable logic learning.

1. W. Dai et al. Bridging Machine Learning and Logical Reasoning by Abductive Learning. NeurIPS. 2019, pp. 2811–2822.

Inductive Logic Programming



They apply inductive logic learning on scene graphs generated by deep learning, to extract explainable rules of predicting the object class labels.

1. Yuan Yang and Le Song. Learn to Explain Efficiently via Neural Logic Inductive Learning. ICLR. 2020.

Agenda

- Network Embedding
- Revisiting Network Embedding
- Graph Neural Network
- Revisiting Graph Neural Network
- GNN&Reasoning
- Revisiting GNN&Reasoning

Challenge: only System 1 DL

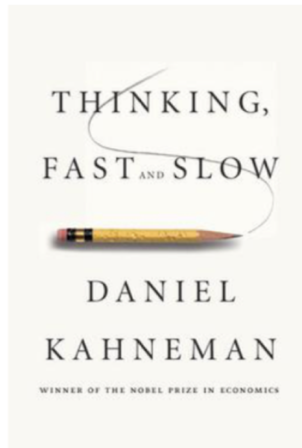
SYSTEM 1 VS. SYSTEM 2 COGNITION

2 systems (and categories of cognitive tasks):

Manipulates high-level / semantic concepts, which can be recombined combinatorially

System 1

- Intuitive, fast, **UNCONSCIOUS**, non-linguistic, habitual
- Current DL



System 2

- Slow, logical, sequential, **CONSCIOUS**, linguistic, algorithmic, planning, reasoning
- Future DL

Mila

3

1. From Bengio's NIPS'2019 Keynote

认知图谱(System 2 DL)：算法与认知的结合

Question: Who is the director of the 2003 film which has scenes in it filmed at the Quality Cafe in Los Angeles?

Quality Café

The Quality Cafe is a now-defunct diner in Los Angeles, California. The restaurant has appeared as a location featured in a number of Hollywood films, including Old School, Gone in 60 Seconds, ...

Los Angeles

Los Angeles is the most populous city in California, the second most populous city in the United States, after New York City, and the third most populous city in North America.

Alessandro Moschitti

Alessandro Moschitti is a professor of the CS Department of the University of Trento, Italy. He is currently a Principal Research Scientist of the Qatar Computing Research Institute (QCRI)

WIKIPEDIA
The Free Encyclopedia

Old School

Old School is a 2003 American comedy film released by Dream Works Pictures and The Montecito Picture Company and directed by Todd Phillips.

Todd Phillips

Todd Phillips is an American director, producer, screenwriter, and actor. He is best known for writing and directing films, including Road Trip (2000), Old School (2003), Starsky & Hutch (2004), and The Hangover Trilogy.

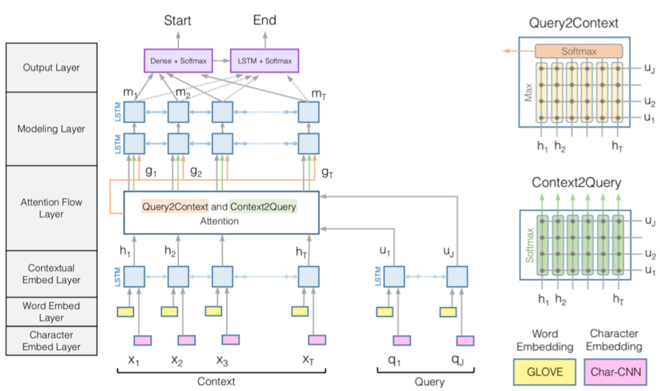
Tsinghua University

Tsinghua University is a major research university in Beijing and dedicated to academic excellence and global development. Tsinghua is perennially ranked as one of the top academic institutions in China, Asia, and worldwide...

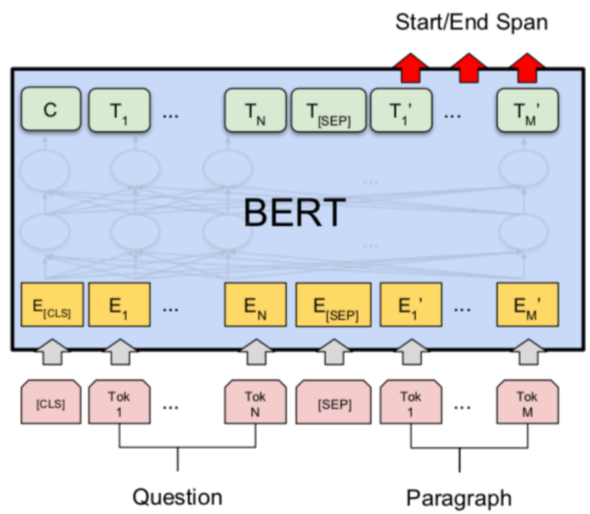
算法：BIDAF, BERT, XLNet

- 目标：理解整个文档，而不仅仅是局部片段
- 但仍然缺乏在知识层面上的**推理能力**

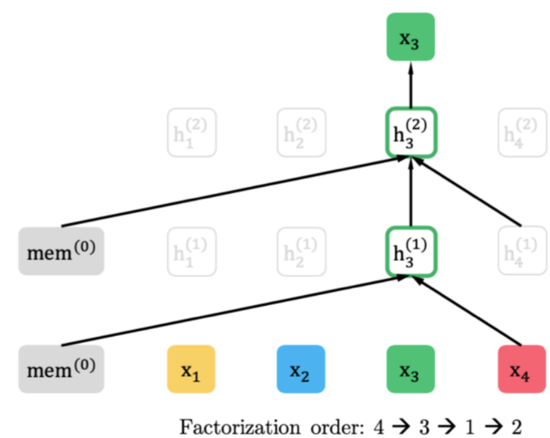
BiDAF



BERT



XLNet

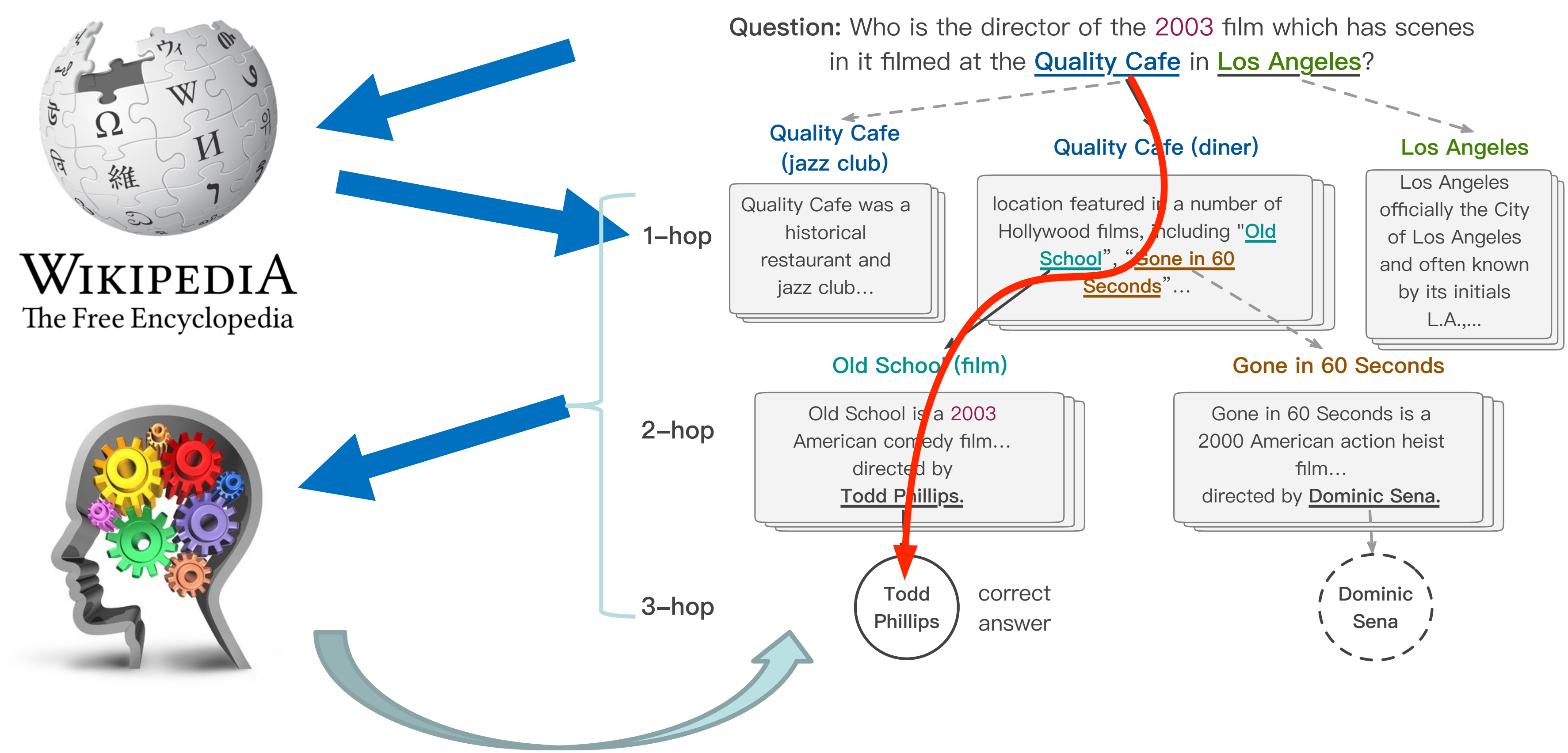


挑战：可解释性

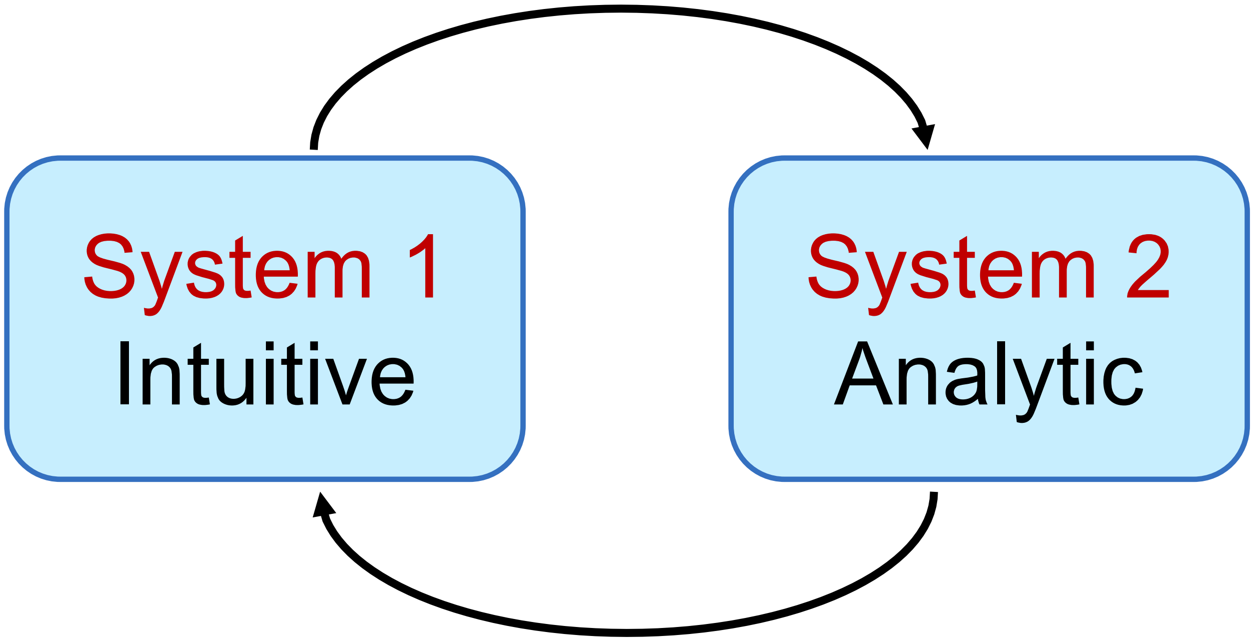
- 大部分阅读理解方法都只能看做**黑盒**：
  - 输入： 问题和文档
  - 输出： 答案文本块（在文档中的起止位置）

- 如何让用户可以验证答案的对错：
  - 推理路径或者子图
  - 每个推理节点上的支撑事实
  - 用于对比的其他可能答案和推理路径

认知图谱：知识表示，推理和决策



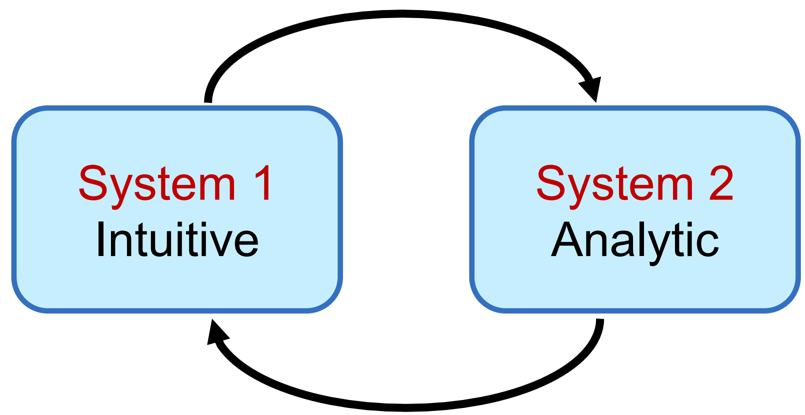
和认知科学的结合



Dual Process Theory (Cognitive Science)

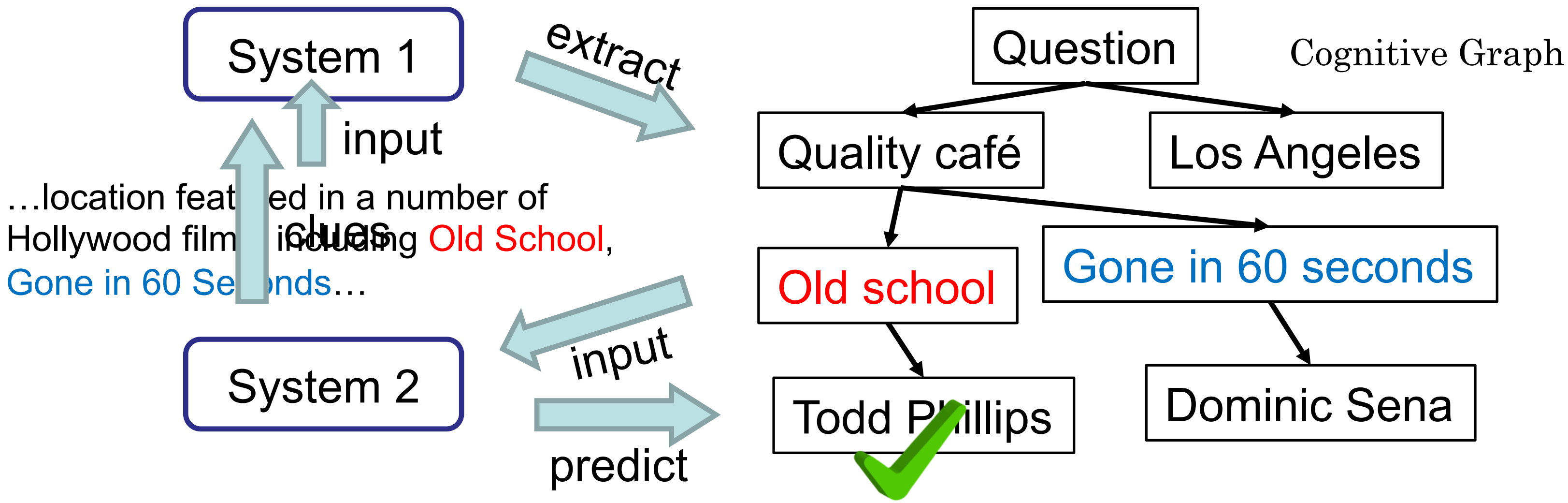
Reasoning w/ Cognitive Graph

- System 1:
  - Knowledge expansion by association in text when reading
- System 2:
  - Decision making w/ all the information



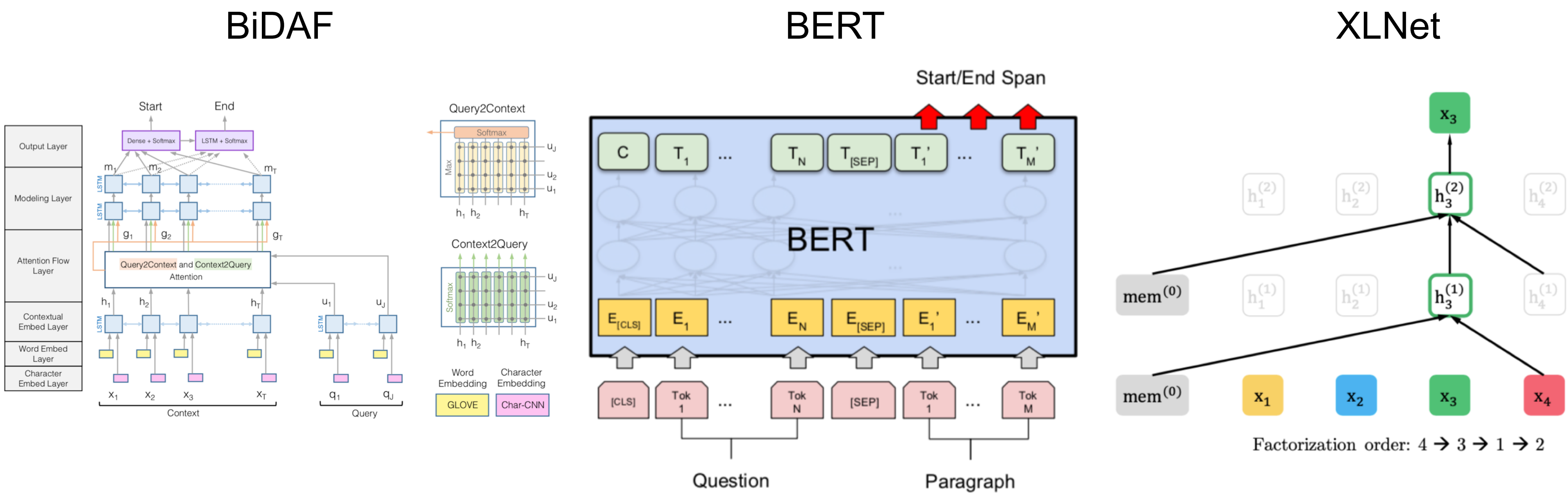
CogQA: Cognitive Graph for QA

- An iterative framework corresponding to dual process theory
- System 1
  - extract entities to build the cognitive graph
  - generate semantic vectors for each node
- System 2
  - Do reasoning based on semantic vectors and graph
  - Feed clues to System 1 to extract next-hop entities



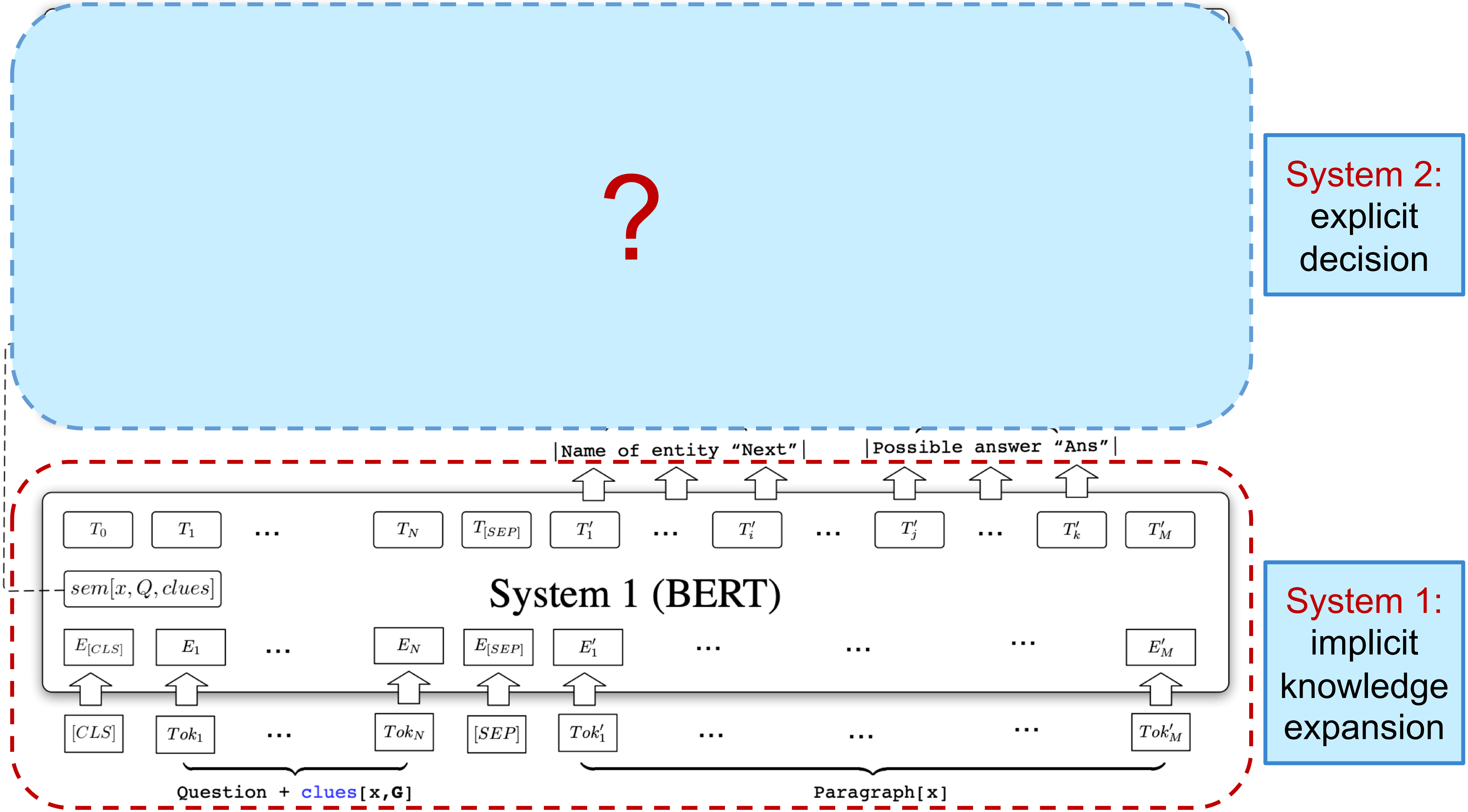
System 1: BIDAF, BERT

- reading comprehension: target at understanding the whole paragraph



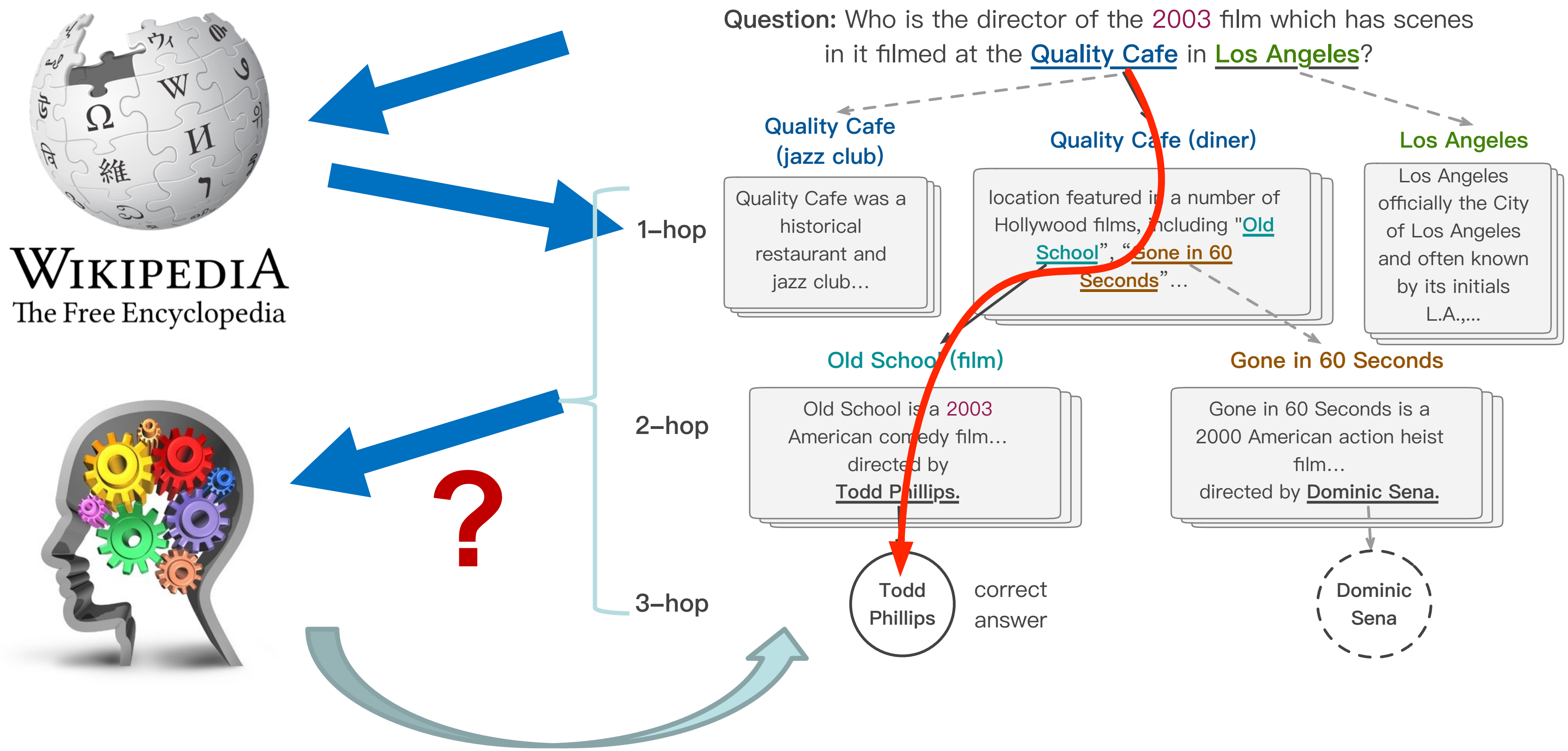
1. Rajpurkar, Pranav, et al. "SQuAD: 100,000+ Questions for Machine Comprehension of Text." EMNLP. 2016.

Cognitive Graph: DL + Dual Process Theory

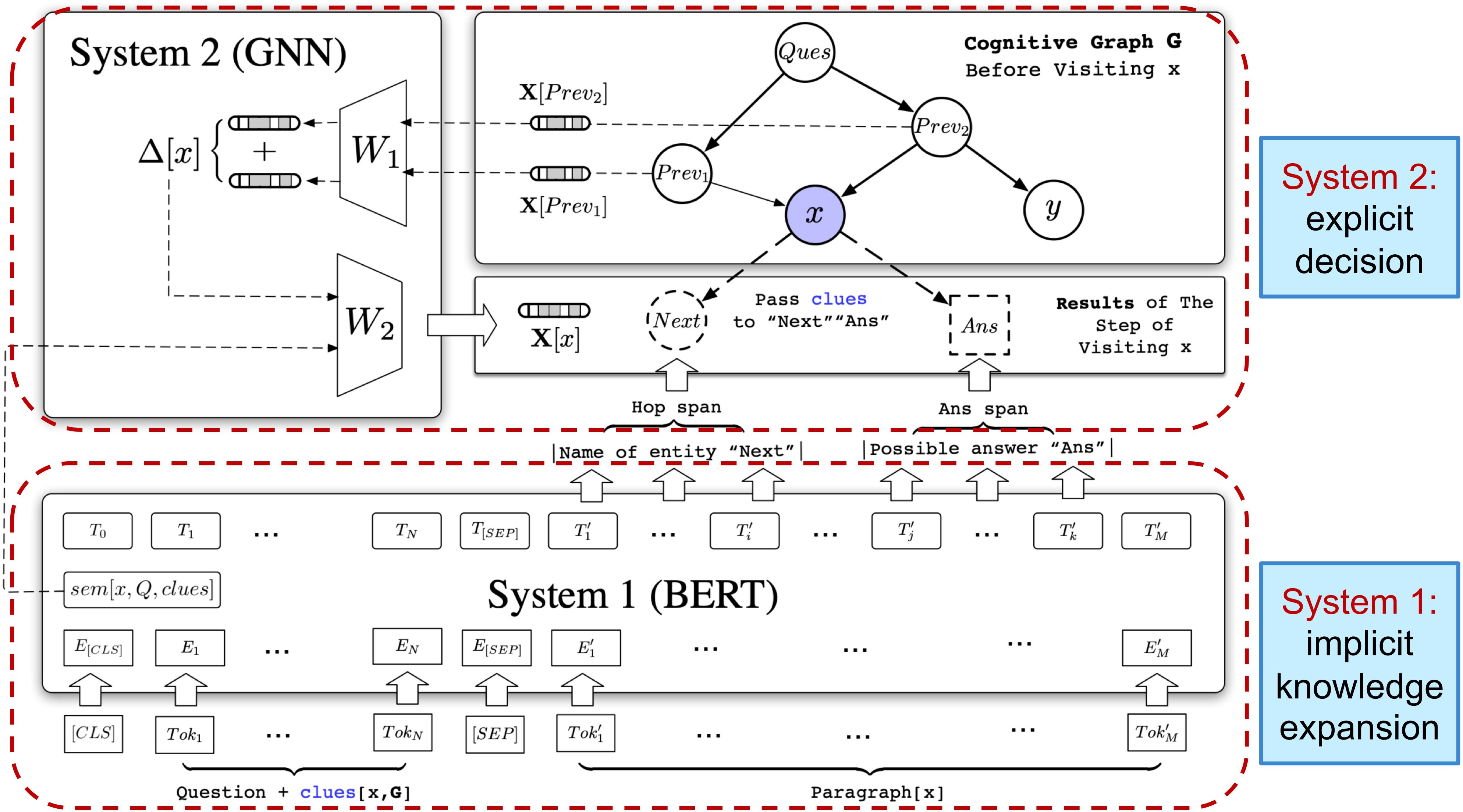


1. M. Ding, C. Zhou, Q. Chen, H. Yang, and J. Tang. Cognitive Graph for Multi-Hop Reading Comprehension at Scale. ACL'19.

System 2: Reasoning, and Decision

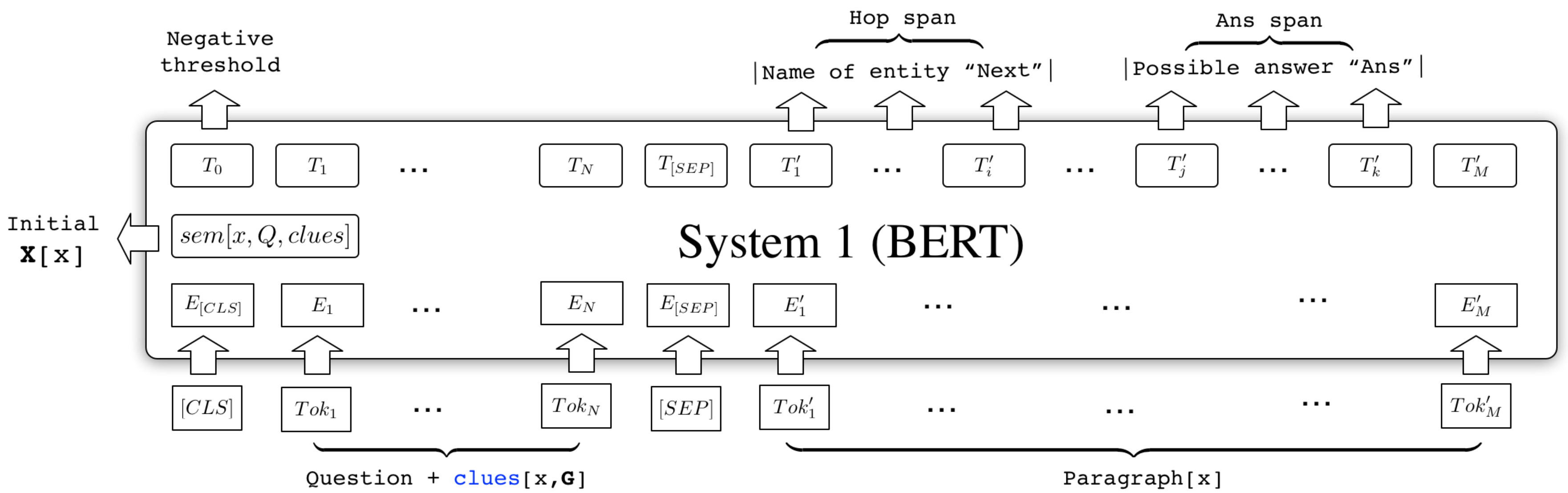


Cognitive Graph: DL + Dual Process Theory



1. M. Ding, C. Zhou, Q. Chen, H. Yang, and J. Tang. Cognitive Graph for Multi-Hop Reading Comprehension at Scale. ACL'19.

System 1: the BERT Implementation



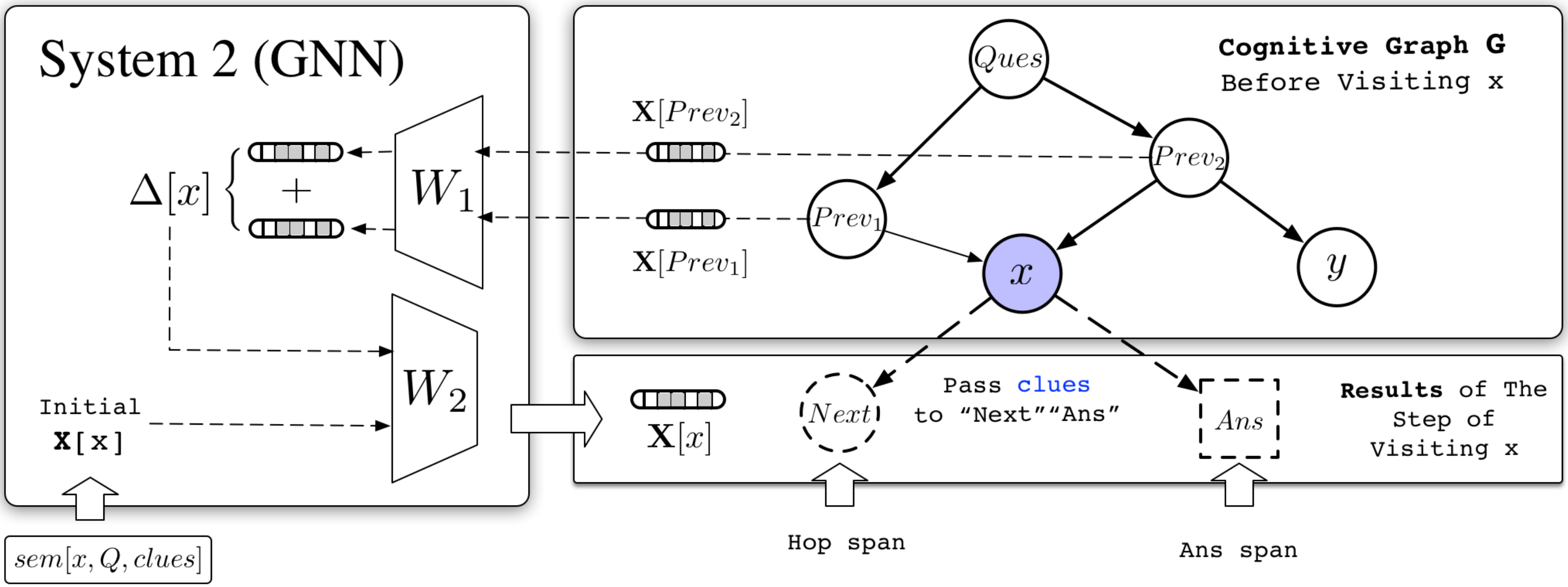
$$\underbrace{[CLS]\ Question\ [SEP]\ clues[x,\mathcal{G}]\ \ [SEP]}_{Sentence\ A}\ \ \underbrace{Para[x]}_{Sentence\ B}$$

$$P_{ans}^{start}[i] = \frac{e^{\mathbf{S}_{ans}\cdot\mathbf{T}_i}}{\sum_j e^{\mathbf{S}_{ans}\cdot\mathbf{T}_j}}$$

$$end_k = \arg\max_{start_k \leq j \leq start_k + maxL} P_{ans}^{end}[j]$$

- Extract top-k next-hop entities and answer candidates respectively
  - Predict the start and end probabilities of each position
- Generate semantic vectors for entities based on their documents
- Take the 0-th probability as negative threshold
  - Ignore the spans whose start probabilities are small than the negative threshold

System 2: the GNN implementation



At each step, hidden representations $\mathbf{X}$ for nodes are updated according to the propagation rules:

$$\Delta = \sigma((AD^{-1})^T \sigma(\mathbf{X}W_1))$$
$$\mathbf{X}' = \sigma(\mathbf{X}W_2 + \Delta)$$

Predictor $\mathcal{F}$ is a two-layer MLP, which predicts the final answer based on hidden representations $\mathbf{X}$:

$$answer = \underset{\text{answer node } x}{\arg\max} \ \mathcal{F}(\mathbf{X}[x])$$

Performance

- HotpotQA is a dataset with leaderboard similar to SQuAD
- CogQA ranked 1st from 21, Feb to 15, May (nearly 3 month)

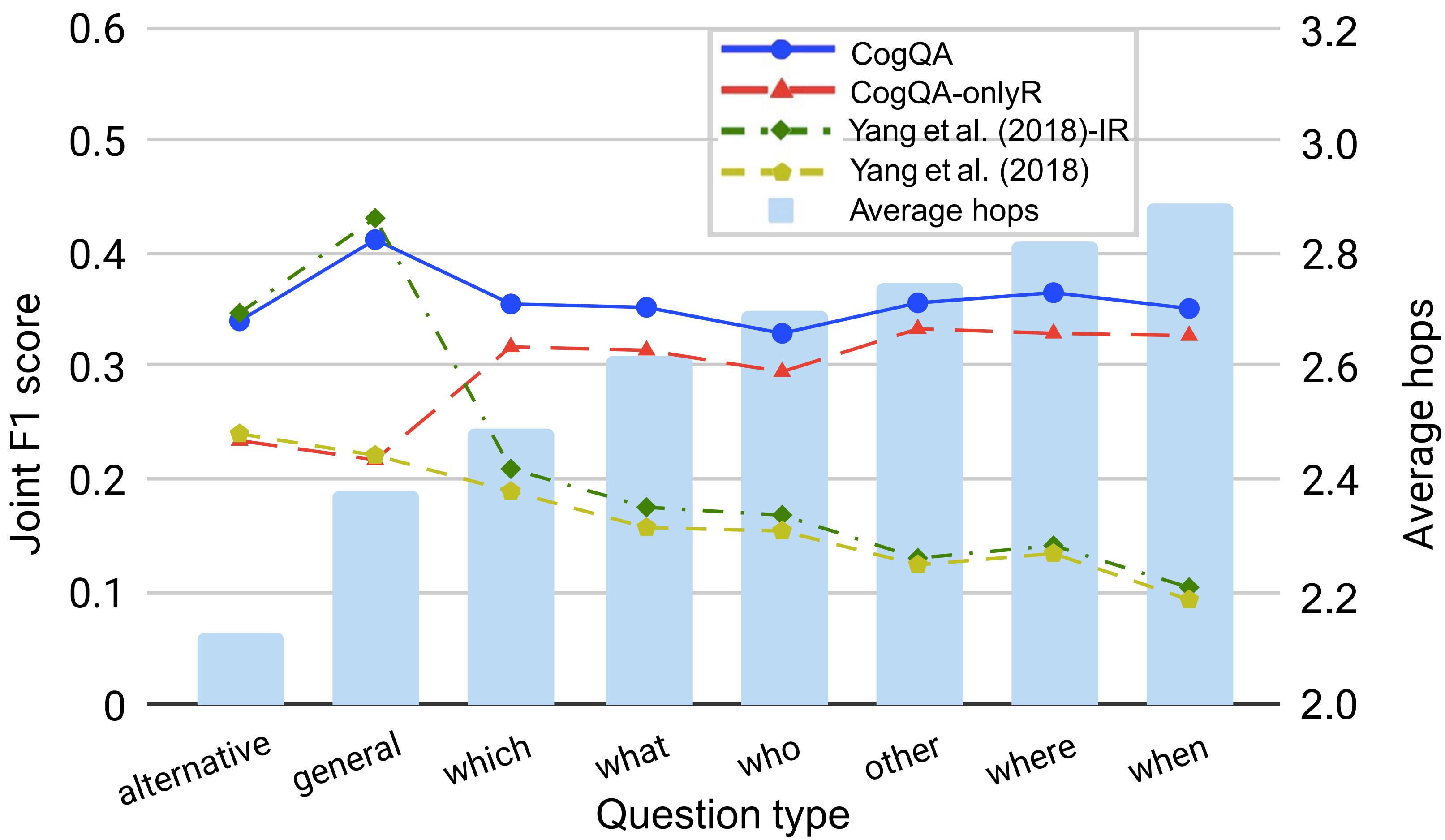
	Model	Ans				Sup				Joint			
		EM	F_1	Prec	Recall	EM	F_1	Prec	Recall	EM	F_1	Prec	Recall
Dev	Yang et al. (2018)	23.9	32.9	34.9	33.9	5.1	40.9	47.2	40.8	2.5	17.2	20.4	17.8
	Yang et al. (2018)-IR	24.6	34.0	35.7	34.8	10.9	49.3	52.5	52.1	5.2	21.1	22.7	23.2
	BERT	22.7	31.6	33.4	31.9	6.5	42.4	54.6	38.7	3.1	17.8	24.3	16.2
	CogQA-sys1	33.6	45.0	47.6	45.4	**23.7**	58.3	**67.3**	56.2	12.3	32.5	39.0	31.8
	CogQA-onlyR	34.6	46.2	48.8	46.7	14.7	48.2	56.4	47.7	8.3	29.9	36.2	30.1
	CogQA-onlyQ	30.7	40.4	42.9	40.7	23.4	49.9	56.5	48.5	**12.4**	30.1	35.2	29.9
	CogQA	**37.6**	**49.4**	**52.2**	**49.9**	23.1	**58.5**	64.3	**59.7**	12.2	**35.3**	**40.3**	**36.5**
Test	Yang et al. (2018)	24.0	32.9	-	-	3.86	37.7	-	-	1.9	16.2	-	-
	QFE	28.7	38.1	-	-	14.2	44.4	-	-	8.7	23.1	-	-
	DecompRC	30.0	40.7	-	-	N/A	N/A	-	-	N/A	N/A	-	-
	MultiQA	30.7	40.2	-	-	N/A	N/A	-	-	N/A	N/A	-	-
	GRN	27.3	36.5	-	-	12.2	48.8	-	-	7.4	23.6	-	-
	CogQA	**37.1**	**48.9**	-	-	**22.8**	**57.7**	-	-	**12.4**	**34.9**	-	-

Table 1: Results on HotpotQA (fullwiki setting). The test set is not public. The maintainer of HotpotQA only offers EM and F_1 for every submission. N/A means the model cannot find supporting facts.

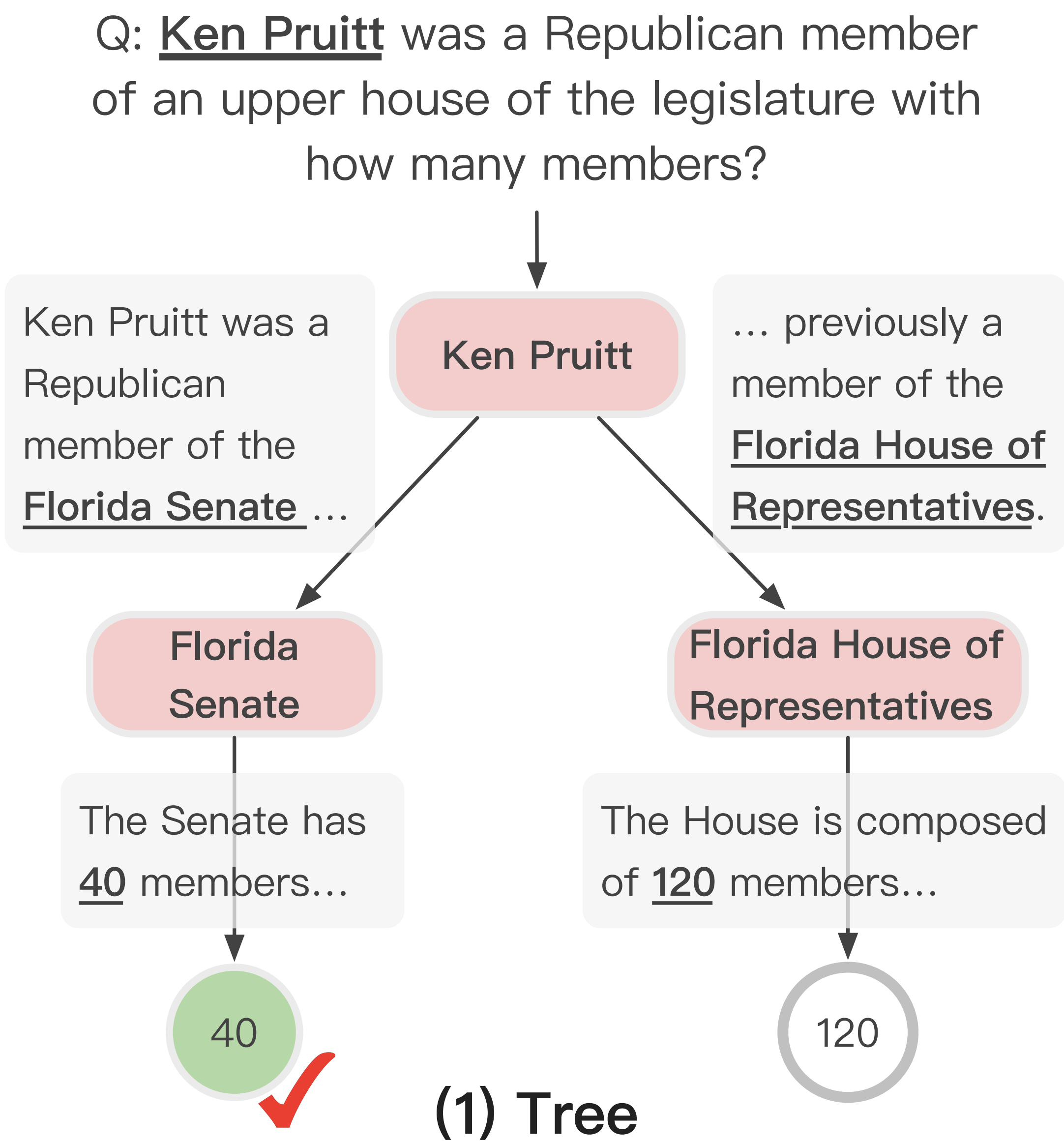
** Code available at https://github.com/THUDM/CogQA

Reasoning Power

CogQA Performs much better on question with more hops !

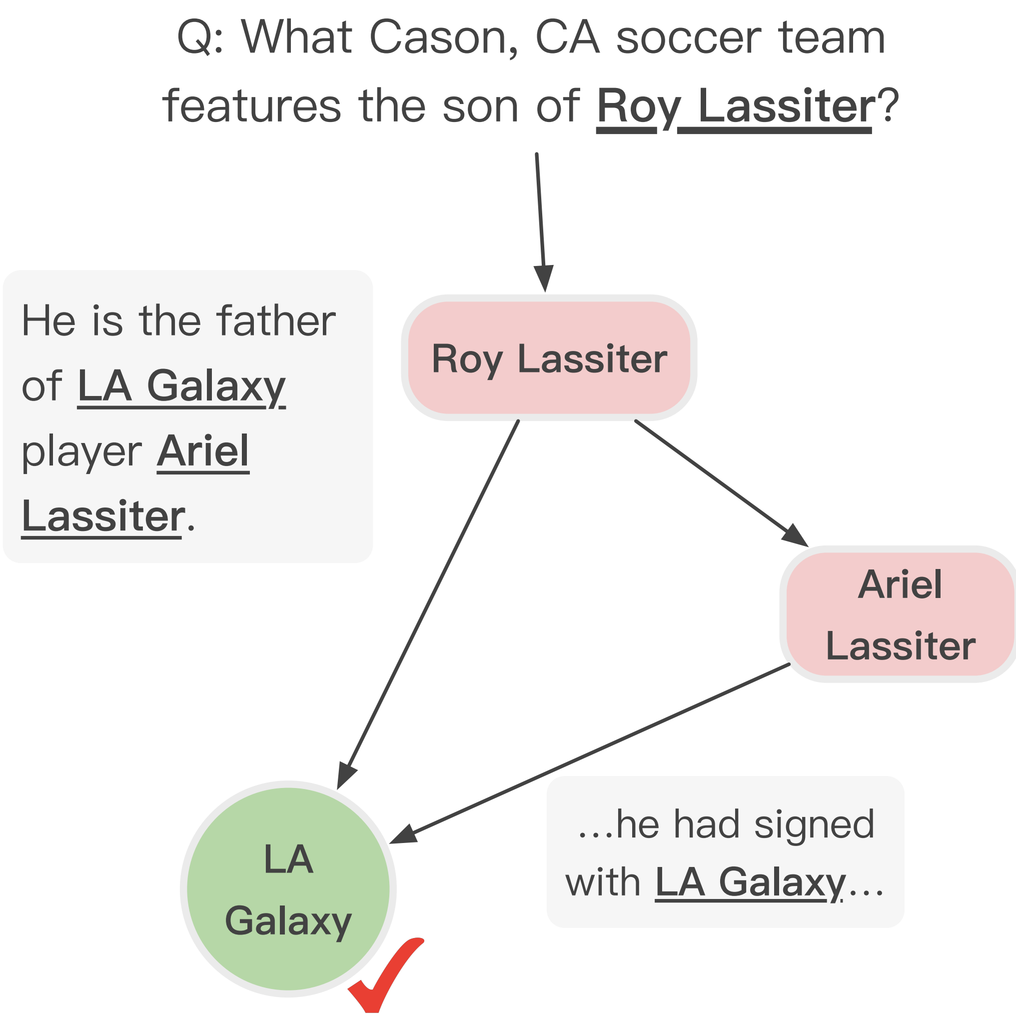


Case Study



- Tree-shape Cognitive Graph
- Users can verify the answer by comparing it with another possible reasoning chain.
- "*Upper House*" in the question is similar to "*Senate*" not "*House of Representative*"

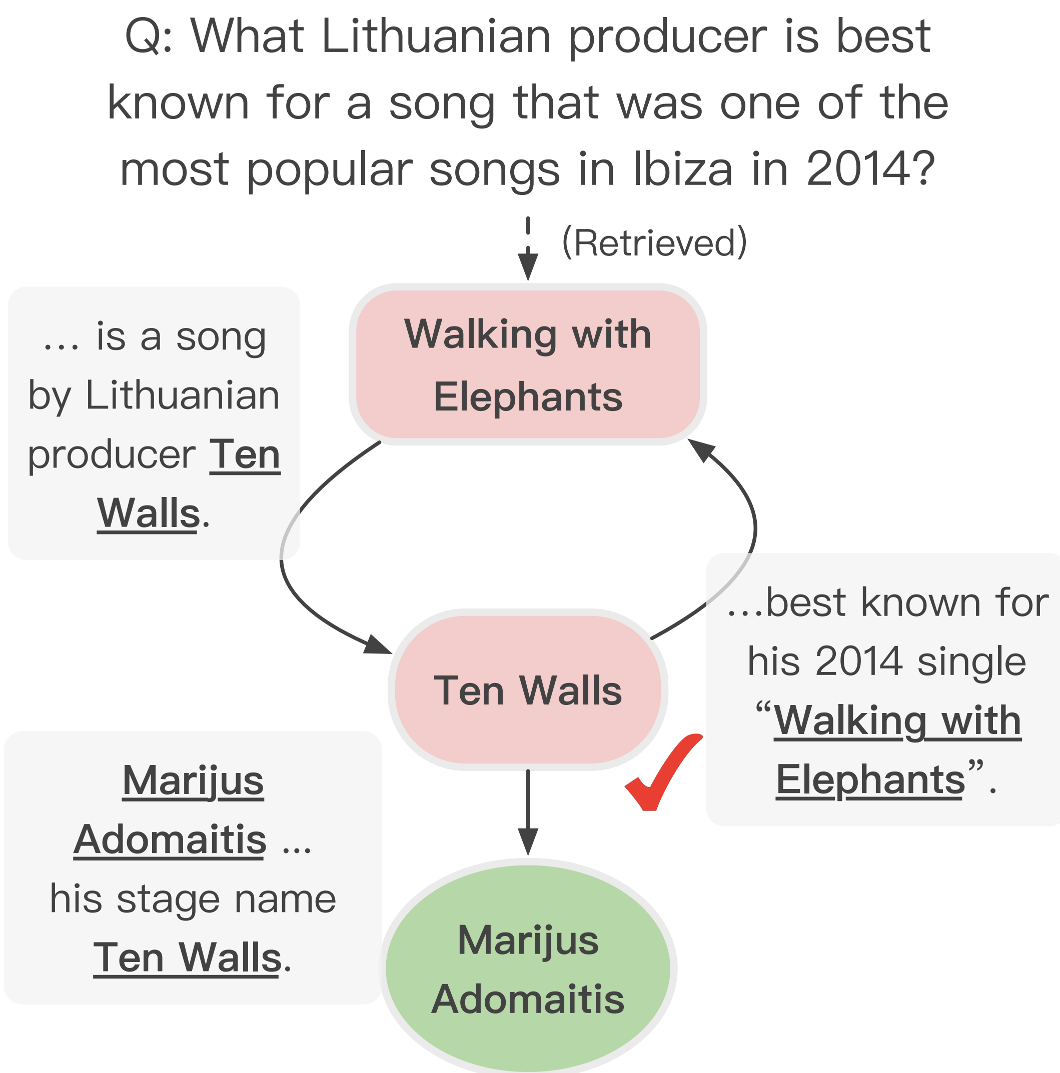
Case Study



(2) DAG

- DAG-shape Cognitive Graph
- Multiple supporting facts provides richer information, increasing the credibility of the answer.

Case Study



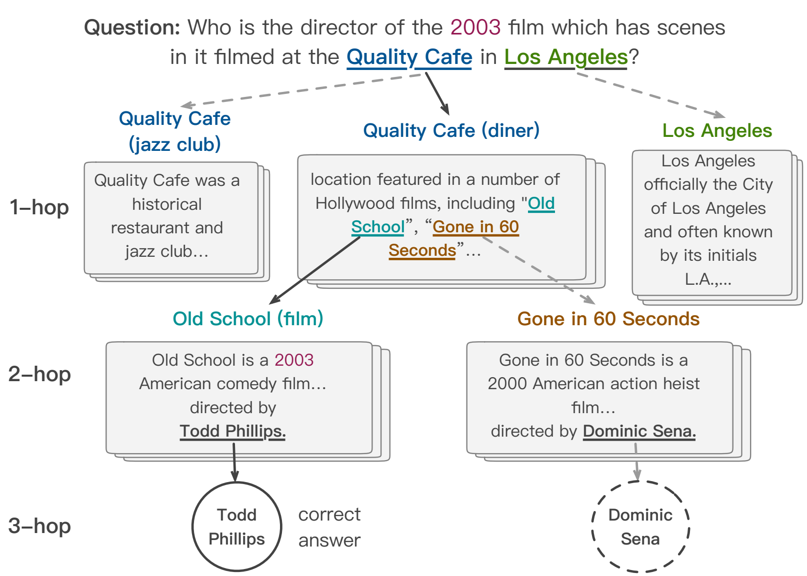
(3) Cyclic Graph

- CogQA gives the answer "Marijus Adomaitis" while the ground truth is "Ten Walls".
- By examining, Ten Walls is just the stage name of Marijus Adomaitis!
- Without cognitive graphs, black-box models cannot achieve it.

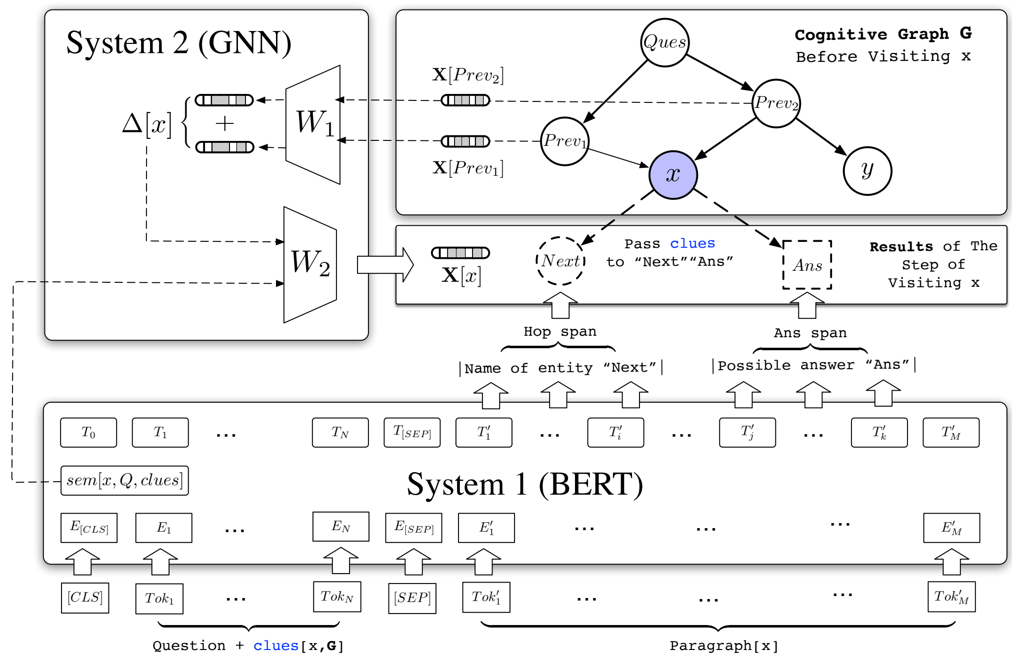
Summary

- Iterative Framework --> Myopic Retrieval
- Cognitive Graph --> Explainability
- Dual process theory --> System 2 Reasoning

Cognitive graph

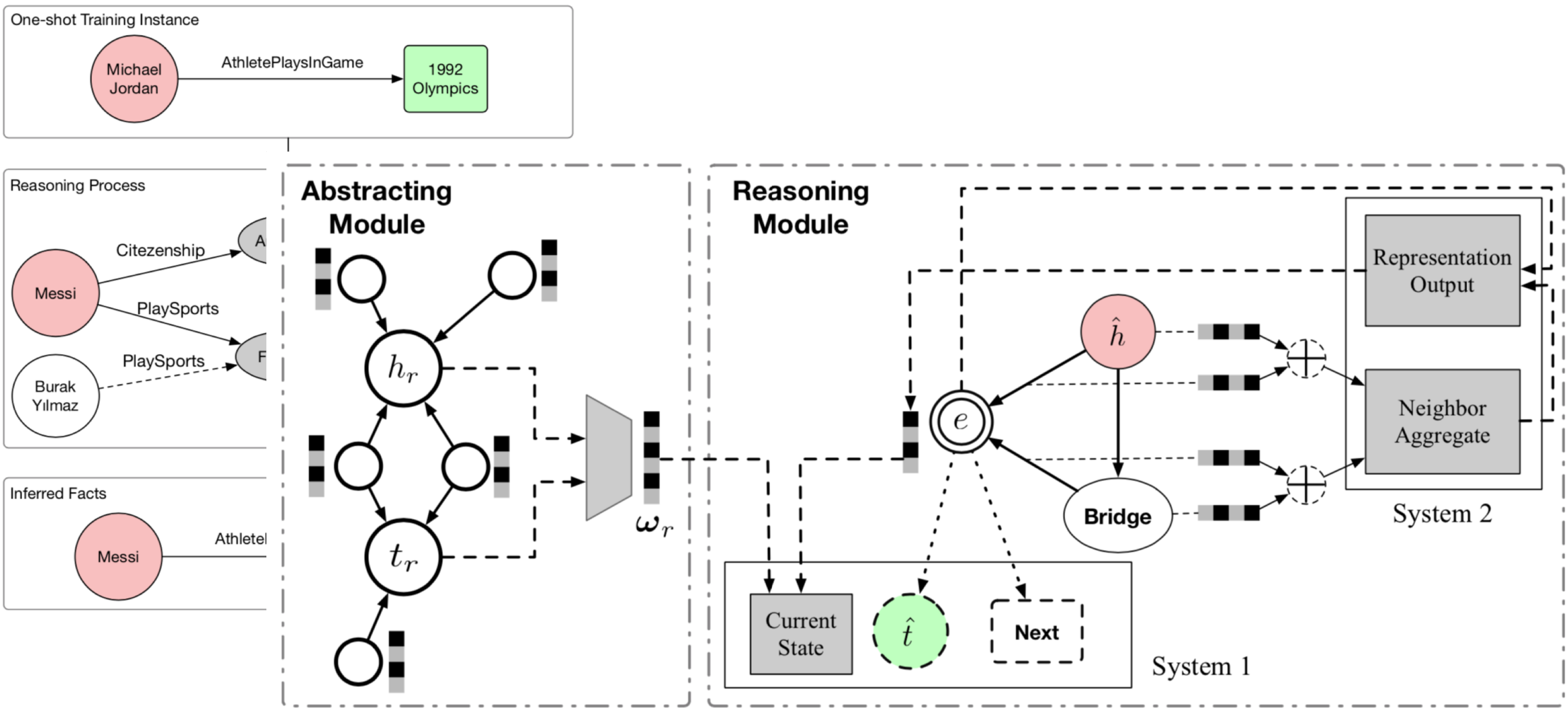


CogQA framework



More Applications: KG completion

- Completing knowledge graph with cognitive graph



Takeaway Messages

- Revisit NE, GNN & Reasoning
  - Network Embedding
  - Graph Neural Network
  - GNN&Reasoning
- Summary
  - Pre-training and Self-supervised learning is becoming more and more important
  - System 2 DL for reasoning and logistical
- What is the Next?

挑战与未来(Next 10)

Edward Feigenbaum
Turing Award Winner

认知与推理

—Trillion-scale common-sense knowledge graph

Tim Berners Lee
Turing Award
Winner

Big Data ⇨ Knowledge ⇨ Intelligence

* AI = Knowledge + Intelligence

1. J. Tang, J. Zhang, L. Yao, J. Li, L. Zhang, and Z. Su. ArnetMiner: Extraction and Mining of Academic Social Networks. KDD'08. pp.990-998.

挑战与未来(Next 30)

意识

—让计算机具有自我意识

认知推理 ⇨ 记忆 ⇨ 自我意识

- Next AI = Reasoning + Memory + Consciousness

1. J. Tang, J. Zhang, L. Yao, J. Li, L. Zhang, and Z. Su. ArnetMiner: Extraction and Mining of Academic Social Networks. KDD’08. pp.990-998.

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 http://keg.cs.tsinghua.edu.cn/jietang
Download all data & Codes https://keg.cs.tsinghua.edu.cn/cogdl/
https://github.com/THUDM