Representation Learning for Networks

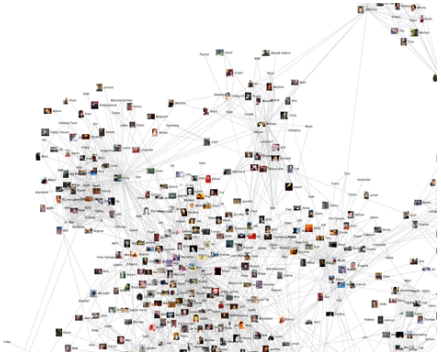
Jie Tang

April 14, 2020

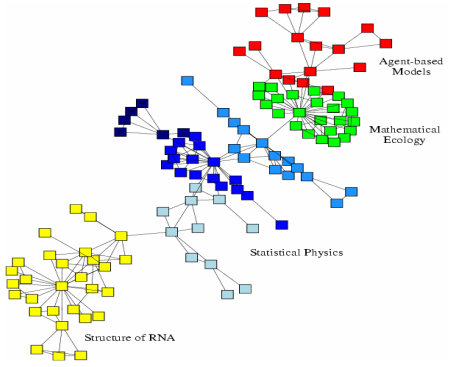
Content

1. Introduction
2. Node Embeddings
3. Graph Neural Networks

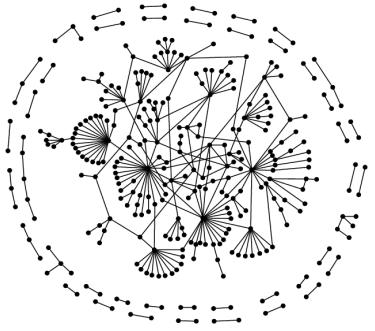
Many Data are Networks[1]



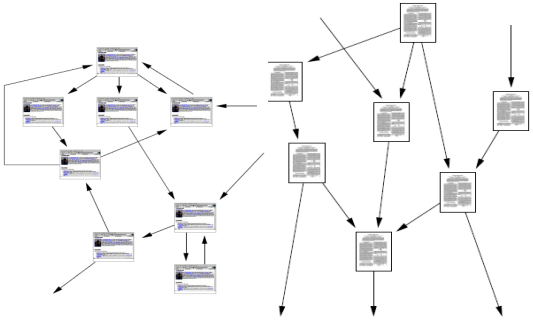
Social networks



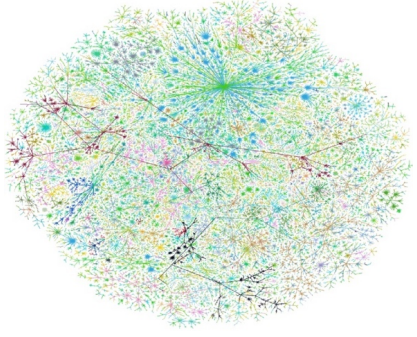
Economic networks



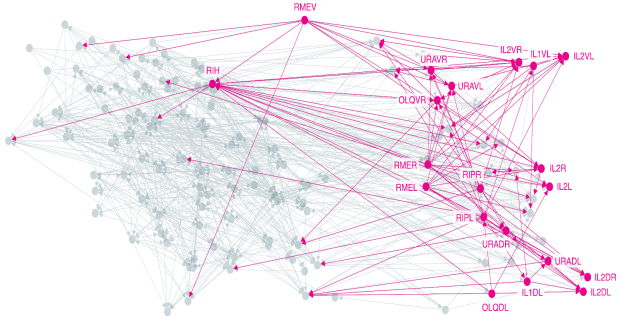
Biomedical networks



Information networks:
Web & citations



Internet



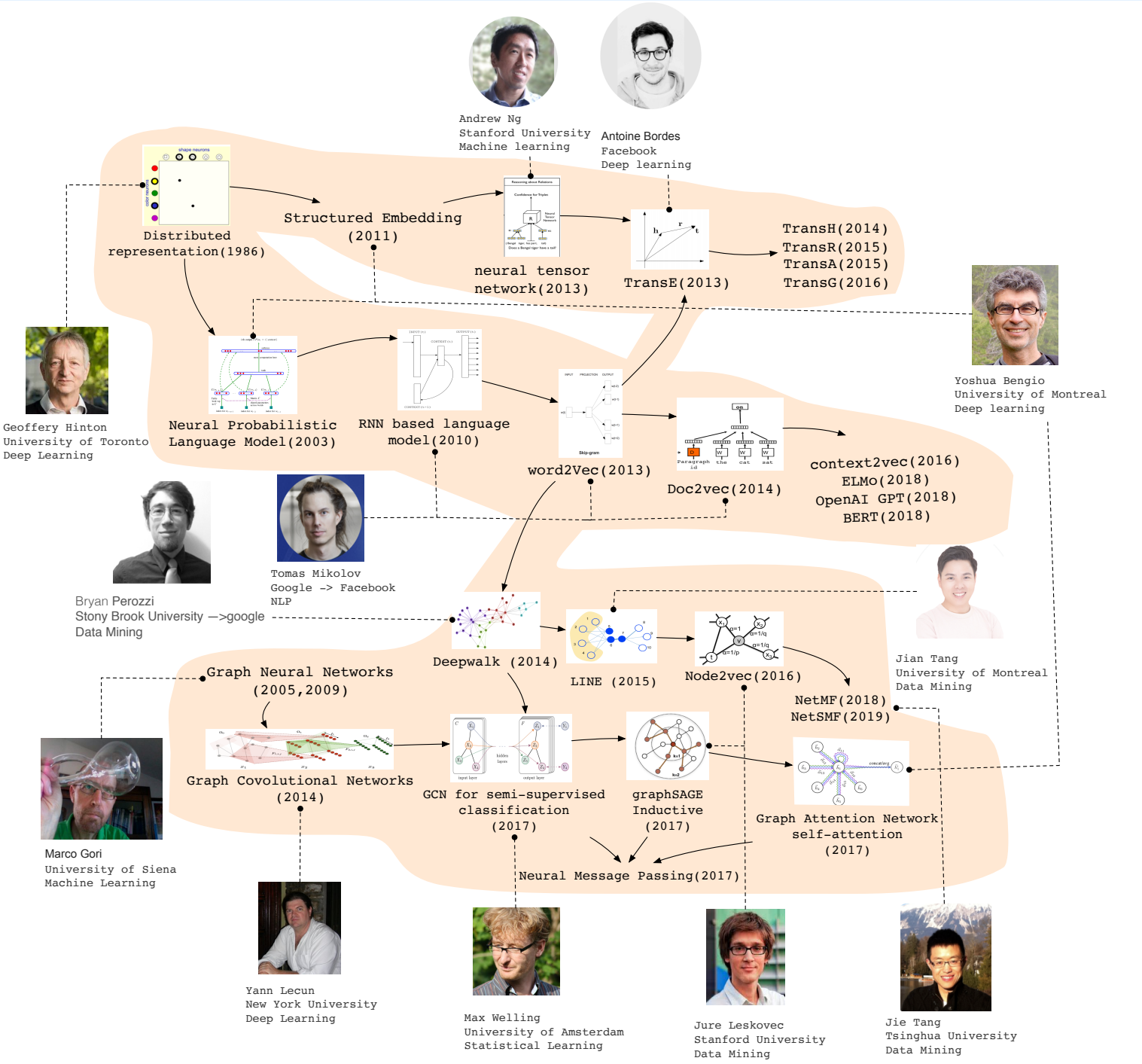
Networks of neurons

[1] Examples from Jure's slides

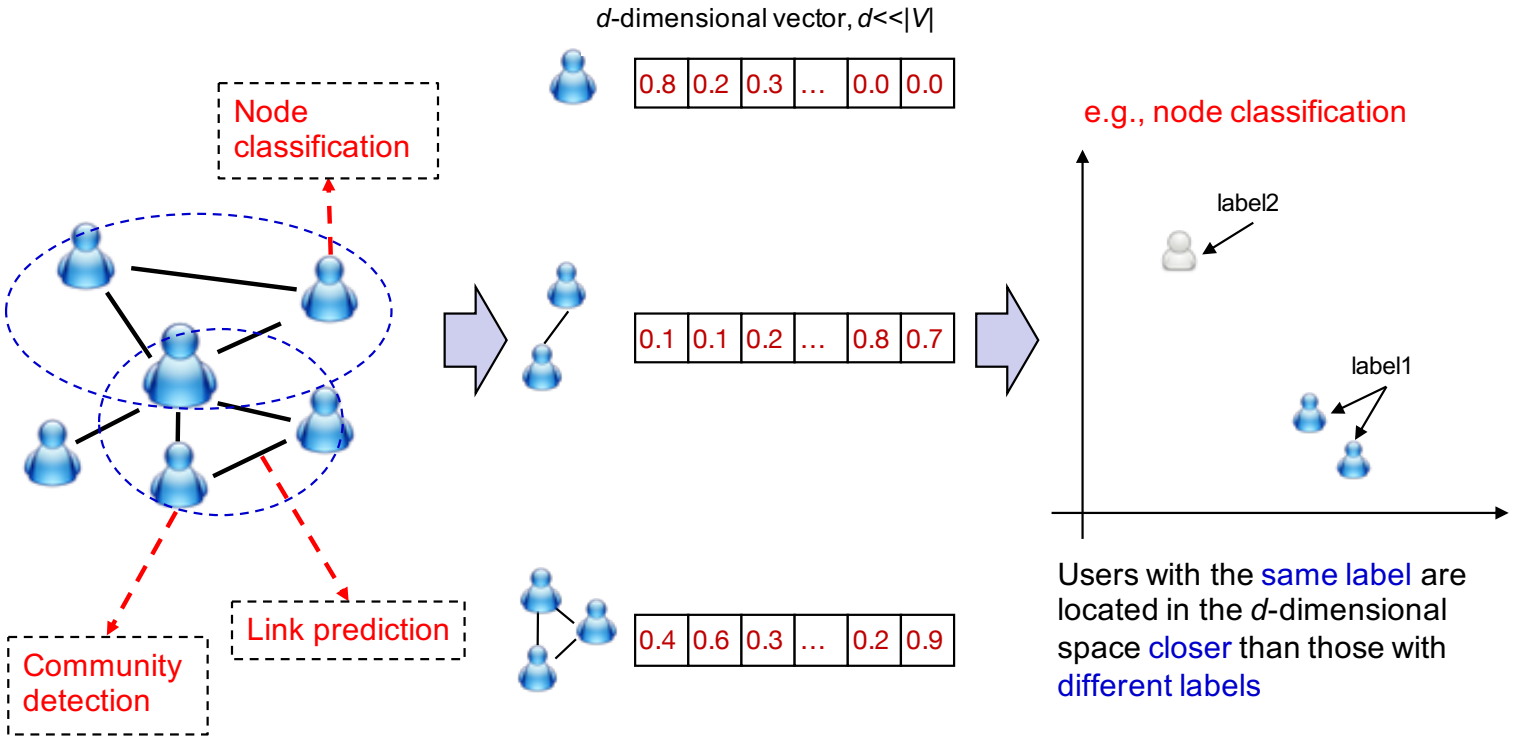
Machine Learning with Networks

Classical 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?


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

Representation Learning for Networks



Why Is It Hard?

Modern deep learning toolbox is designed for simple sequences or grids.

- CNNs for fixed-size images/grids...
- RNNs or word2vec for text/sequences...

But networks are far more complex!

- Complex topographical structure (i.e., no spatial locality like grids)
- No fixed node ordering or reference point (i.e., the isomorphism problem)
- Often dynamic and have multimodal features.

Contents

1 Introduction

2 Node Embeddings

3 Graph Neural Networks

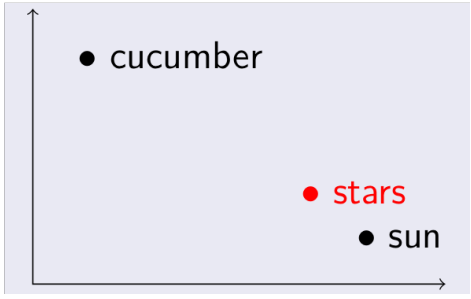
Recall Word Embeddings

- Learning a representation for words:

$$\phi : v \in V \rightarrow R^d$$

```
he curtains open and the stars shining in on the barely
ars and the cold , close stars " . And neither of the w
rough the night with the stars shining so brightly , it
made in the light of the stars . It all boils down , wr
 surely under the bright stars , thrilled by ice-white
sun , the seasons of the stars ? Home , alone , Jay pla
m is dazzling snow , the stars have risen full and cold
```

	shining	bright	trees	dark	look
stars	38	45	2	27	12

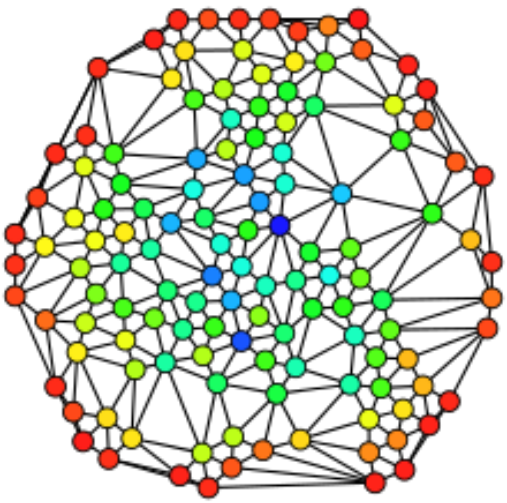


- Ideally, we should have

$$\|\phi(\text{sun}) - \phi(\text{stars})\| < \|\phi(\text{cucumber}) - \phi(\text{stars})\|$$

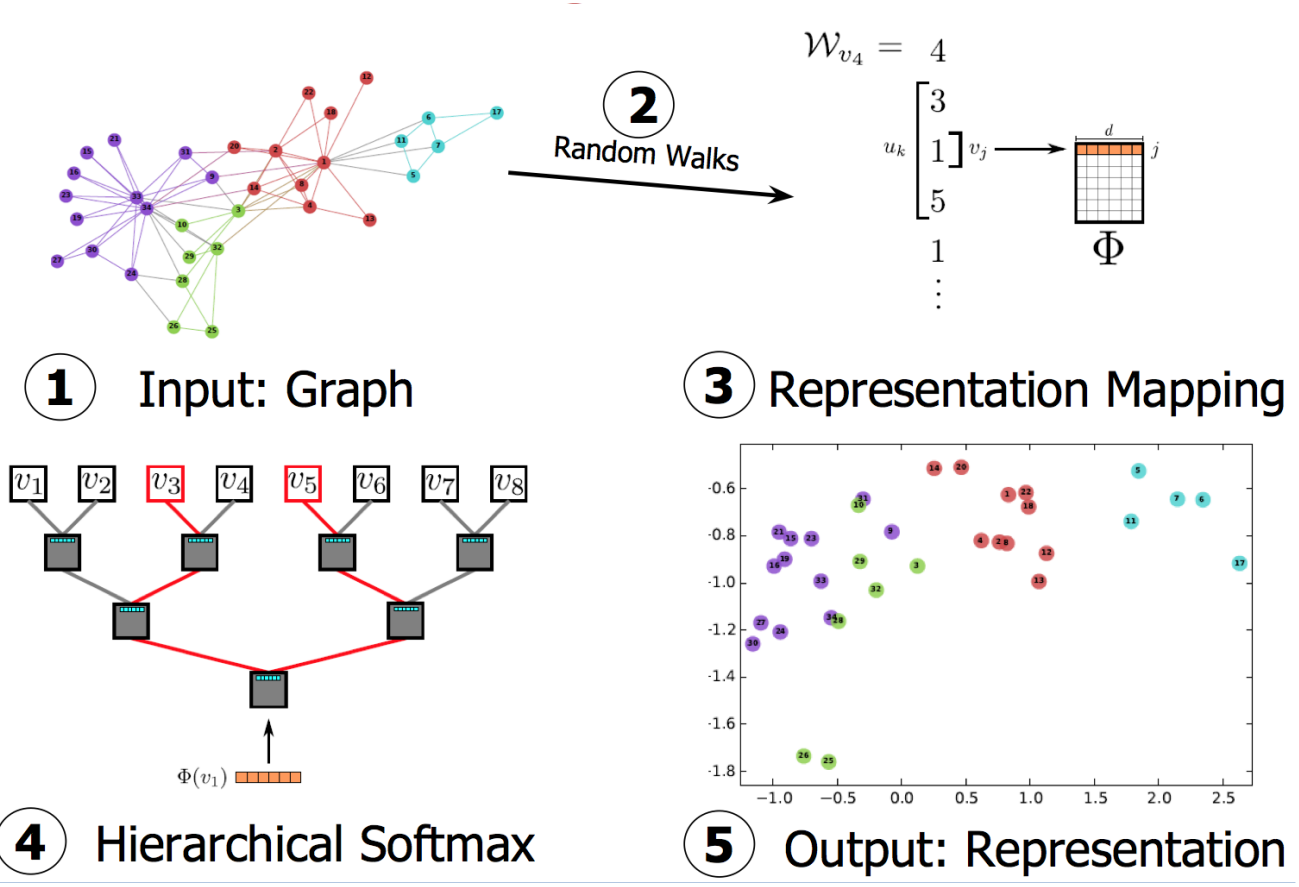
Node Embeddings

- How to learn a representation for each node of networks?



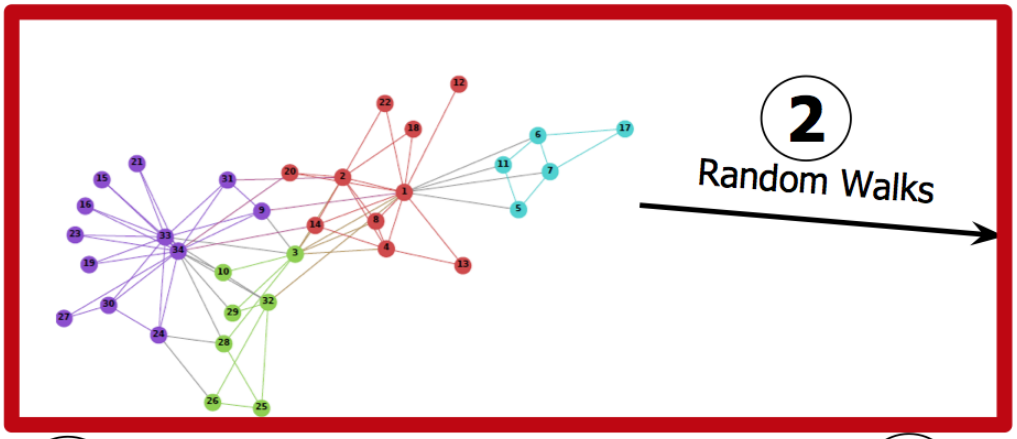
- Input: $G = (V, E)$ or adjacency matrix
- Problem decomposition:
  - how to generate the context for each vertex?
  - how to learn the representation effectively and efficiently?
  - how to incorporate the other unique network properties?
  - how to combine network and content together?

DeepWalk: Learning Network Representations[2]



[2] Perozzi, B., Al-Rfou, R. and Skiena, S., Deepwalk: Online learning of social representations. In KDD'14, pages 701-710

Random Walk

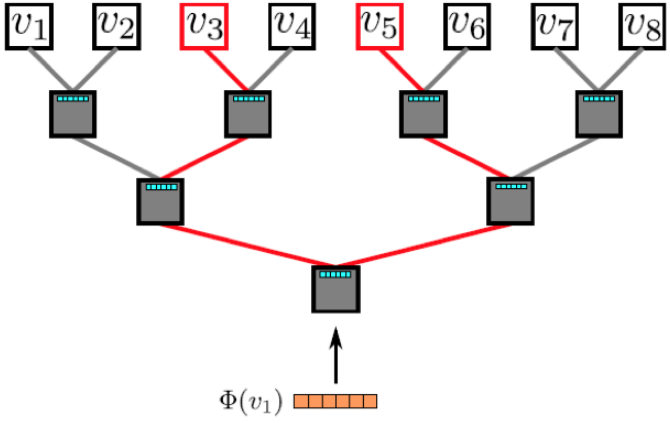


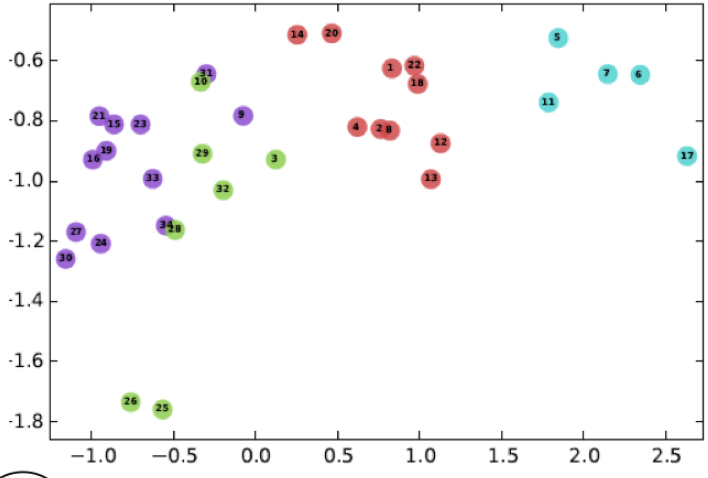
$\mathcal{W}_{v_4} = 4$

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

1 Input: Graph

3 Representation Mapping



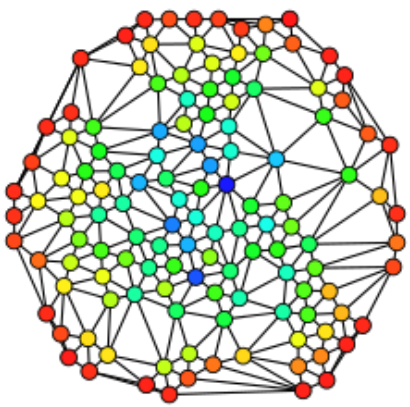


4 Hierarchical Softmax

5 Output: Representation

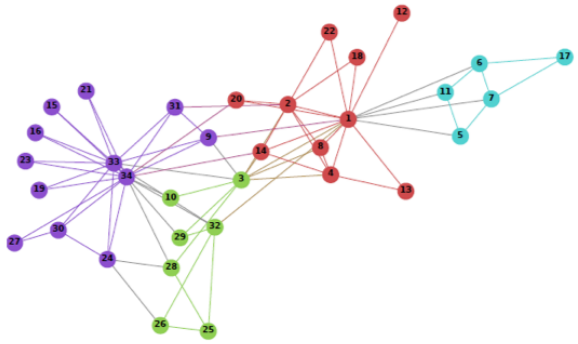
Random Walk

- Generate γ random walks for each vertex
- Each random walk has length t
  - in each random walk step, jump to the next vertex uniformly.
- Example: $v_{46} \rightarrow v_{45} \rightarrow v_{71} \rightarrow v_{24} \rightarrow v_5 \rightarrow v_1 \rightarrow v_{17}$
- Finally, for vertex v_1 in a network, we have

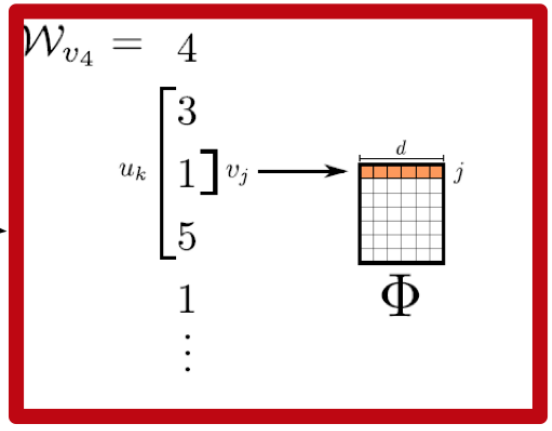


$$
\begin{array}{cccccc}
v_{71} \rightarrow & v_{24} \rightarrow & v_5 \rightarrow & v_1 \rightarrow & v_{17} \rightarrow & v_{80} \rightarrow \\
v_{92} \rightarrow & v_2 \rightarrow & v_3 \rightarrow & v_1 \rightarrow & v_{12} \rightarrow & v_{73} \rightarrow \\
v_{37} \rightarrow & v_{34} \rightarrow & v_9 \rightarrow & v_1 \rightarrow & v_{10} \rightarrow & v_{94} \rightarrow \\
v_{73} \rightarrow & v_{64} \rightarrow & v_5 \rightarrow & v_1 \rightarrow & v_{12} \rightarrow & v_1 \rightarrow \\
v_{75} \rightarrow & v_{14} \rightarrow & v_6 \rightarrow & v_1 \rightarrow & v_{13} \rightarrow & v_{61} \rightarrow
\end{array}
$$

Representation Mapping

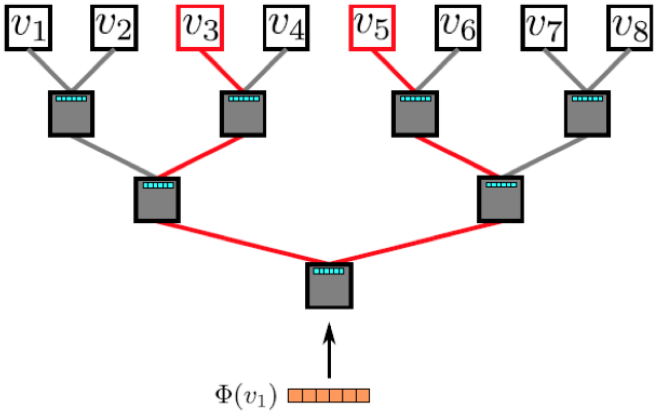


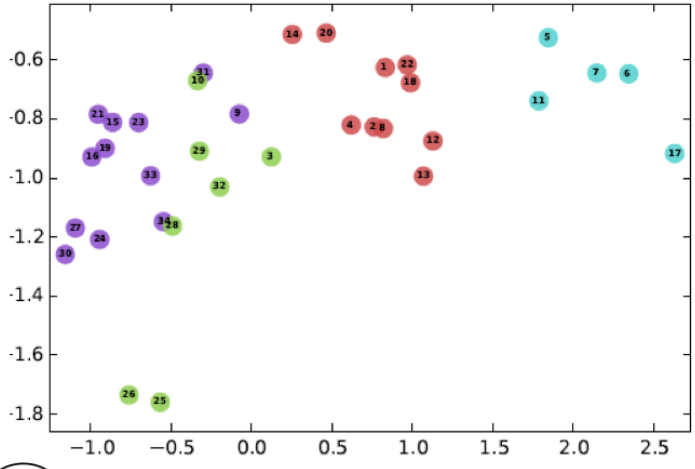
2 Random Walks



1 Input: Graph

3 Representation Mapping



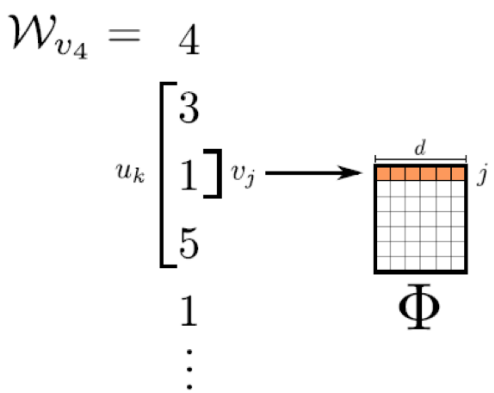


4 Hierarchical Softmax

5 Output: Representation

Representation Mapping

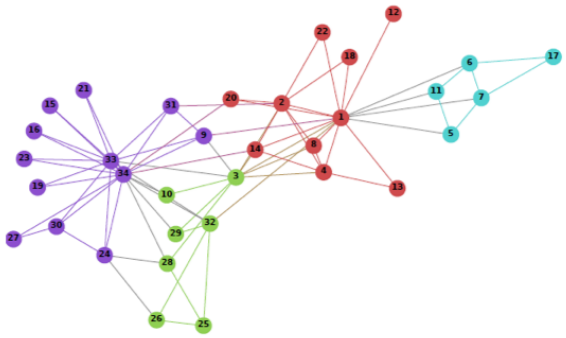
- For a path $\mathcal{W}_{v_4}$: $v_4 \rightarrow v_3 \rightarrow v_1 \rightarrow v_5 \rightarrow v_1 \rightarrow v_{46}$



- Define a window size **w**: if **w** $= 1$ then **v** $= v_1$
- Map the current vertex v_1 to its representation in R^d
- Maximize (skip-gram)

$$P(v_3, v_5 | \phi(v_1))$$

Hierarchical SoftMax



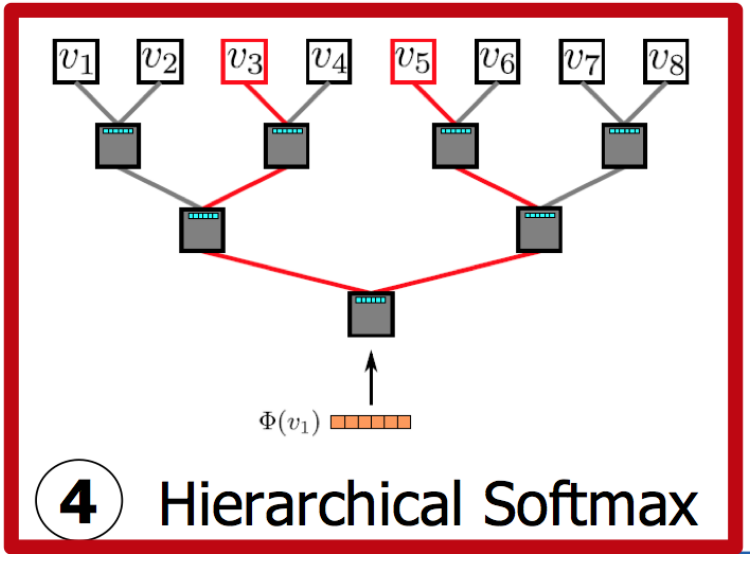
2 Random Walks

$$\mathcal{W}_{v_4} = \begin{matrix} 4 \\ \begin{bmatrix} 3 \\ 1 \\ 5 \end{bmatrix} \\ 1 \\ \vdots \end{matrix}$$

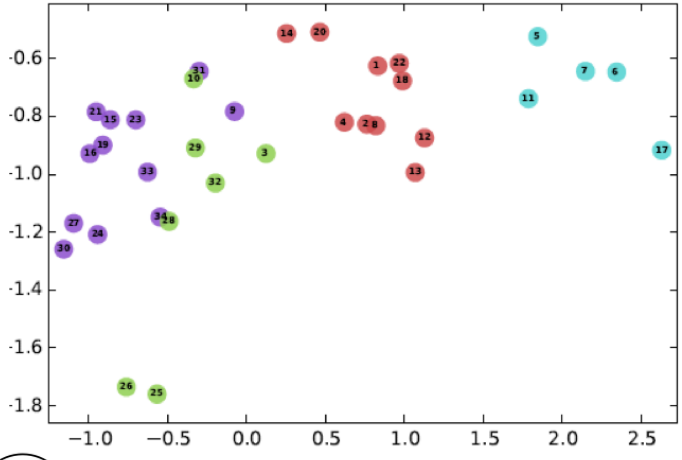
u_k v_j Φ d j

1 Input: Graph

3 Representation Mapping

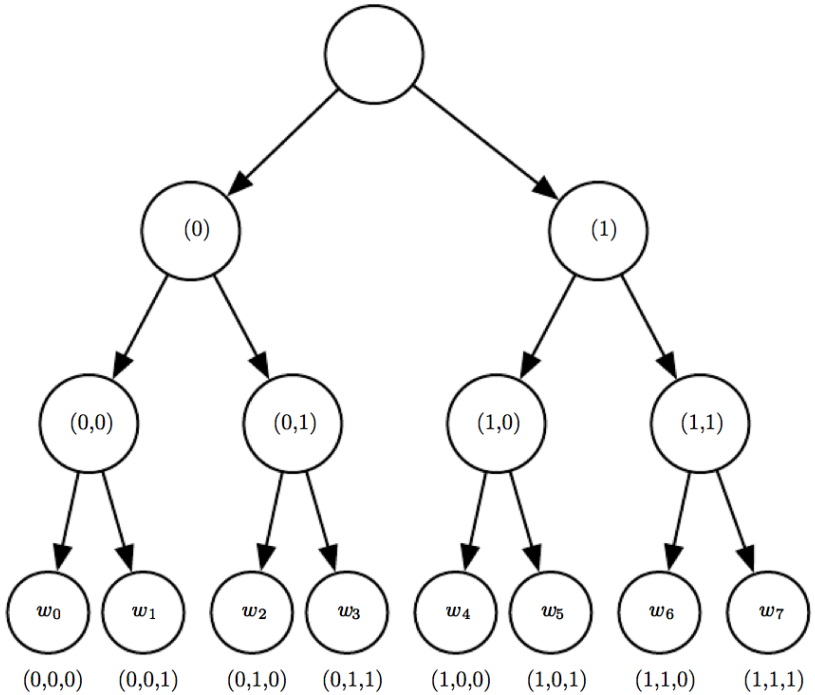


4 Hierarchical Softmax



5 Output: Representation

Hierarchical Softmax



- Then

$$p(v_i|C_i) = \prod_{k=1}^{K} p(d_k|q_k, C_i) = \prod_{k=1}^{K} \left(\sigma(q_k \cdot C_i)^{1-d_k}(1-\sigma(q_k \cdot C_i))^{d_k}\right)$$

where σ is the sigmoid function, q_k is the vector of non-leaf node on the path from root to word leaf, d_k is the corresponding code of q_k.

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.

Experimental Results

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 multi-label classification
- DeepWalk (vertex representation learning) performs well, especially when labels are sparse.

DeepWalk Results

Results: YouTube

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

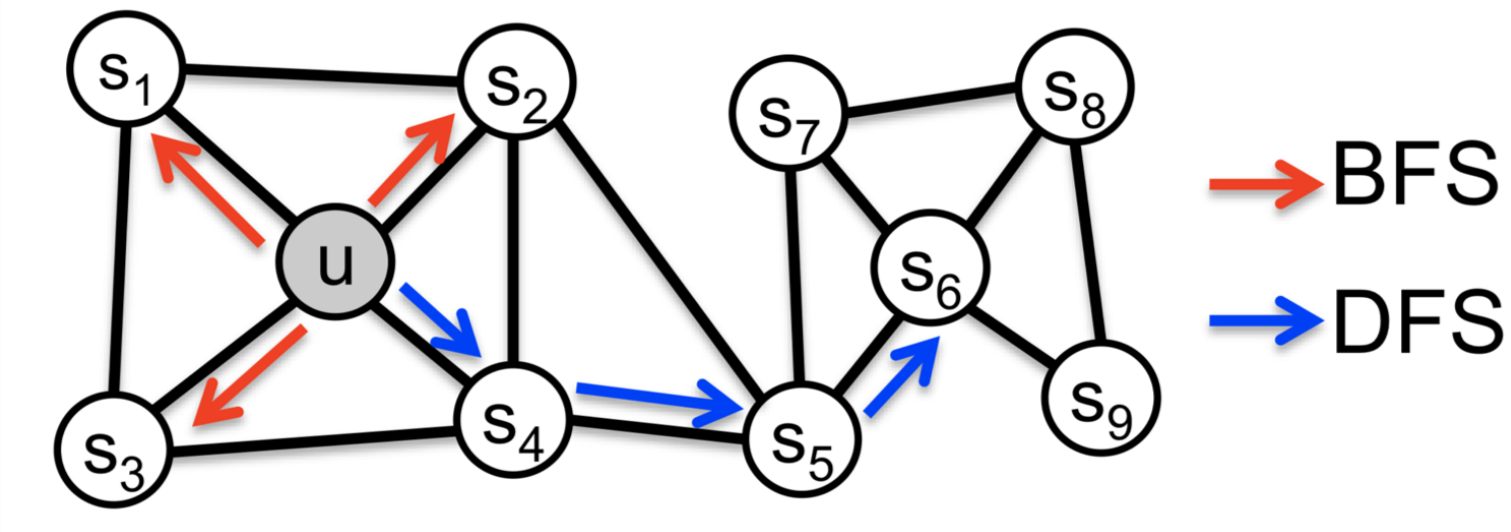
	% Labeled Nodes	1%	2%	3%	4%	5%	6%	7%	8%	9%	10%
Micro-F1(%)	DeepWalk	**37.95**	**39.28**	**40.08**	**40.78**	**41.32**	**41.72**	**42.12**	**42.48**	**42.78**	**43.05**
	SpectralClustering	—	—	—	—	—	—	—	—	—	—
	EdgeCluster	23.90	31.68	35.53	36.76	37.81	38.63	38.94	39.46	39.92	40.07
	Modularity	—	—	—	—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

node2vec: Biased Walks[3]

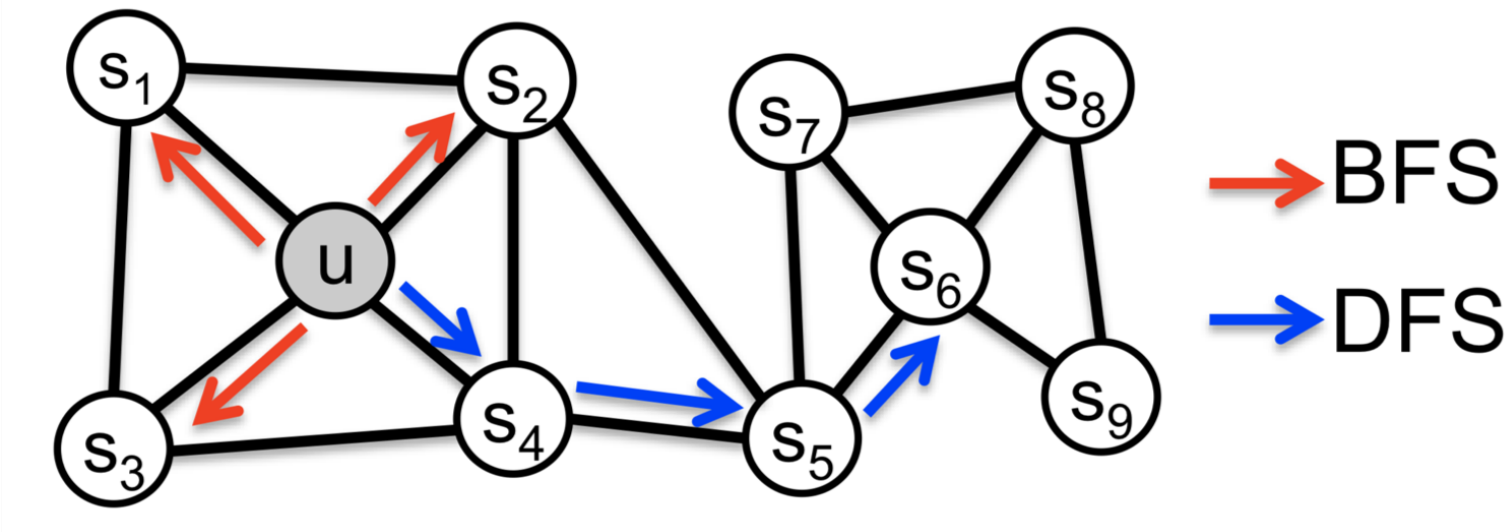
Idea: use flexible, biased random walks that can trade off between local and global views of the network.



[3] Grover A, Leskovec J. node2vec: Scalable feature learning for networks[C]//Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2016: 855-864.

node2vec:Biased Walks

Two classic strategies to define a neighborhood $N_R(u)$ of a given node u:



- $N_{BFS}(u) = \{s_1, s_2, s_3\}$ Local microscopic view
- $N_{DFS}(u) = \{s_1, s_2, s_3\}$ Global macroscopic view

Biased Random Walks

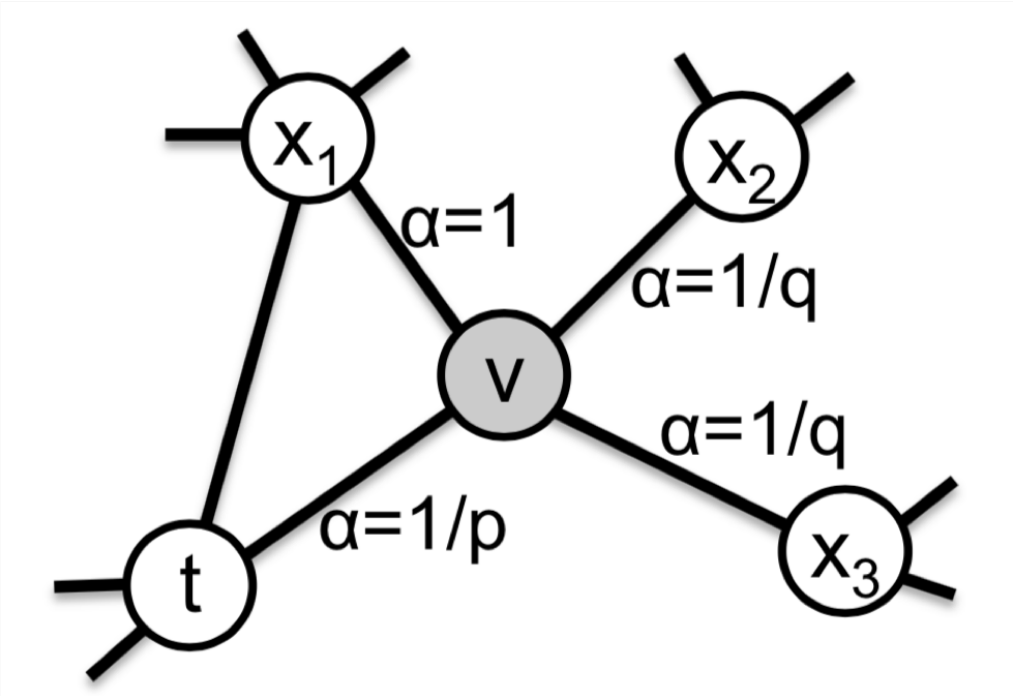
Biased random walk R that given a node u generates neighborhood $N_R(u)$

- Two parameters:
  - Return parameter p: Return back to the previous node.
  - In-out parameter q: Moving outwards (DFS) vs inwards (BFS)
  - Consider a random walk that just traversed edge (t, v) and now resides at node v. The transition probability from v to the next node x is defined as:

$$\alpha_{pq}(t,x) = \begin{cases} \frac{1}{p} & \text{if } d_{tx} = 0 \\ 1 & \text{if } d_{tx} = 1 \\ \frac{1}{q} & \text{if } d_{tx} = 2 \end{cases}$$

Biased Random Walks

Walker is at v, where to go next?



- BFS-like walk: Low value of p
- DFS-like walk: Low value of q

LINE: Information Network Embedding[4]

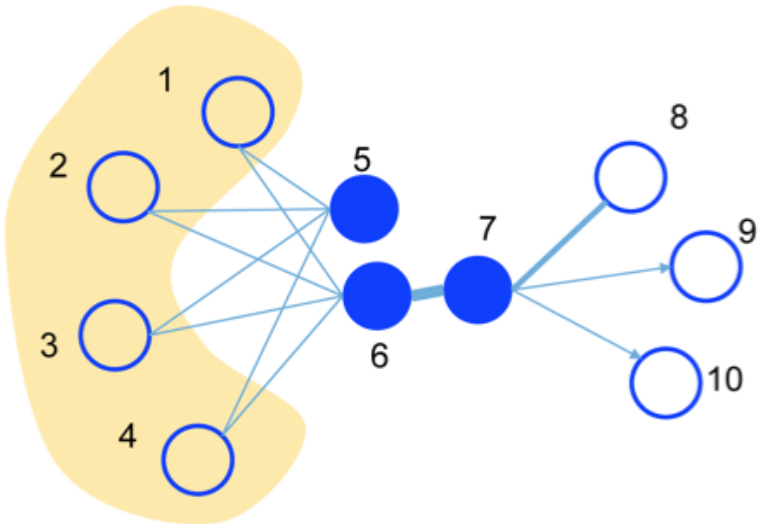


Figure 1: A toy example of information network. Edges can be undirected, directed, and/or weighted. Vertex 6 and 7 should be placed closely in the low-dimensional space as they are connected through a strong tie. Vertex 5 and 6 should also be placed closely as they share similar neighbors.

[4] Tang, J., Qu, M., Wang, M., Zhang, M., Yan, J. and Mei, Q., 2015, May. Line: Large-scale information network embedding. In WWW'15, pages 1067-1077.

Line: First-order Proximity

Definition

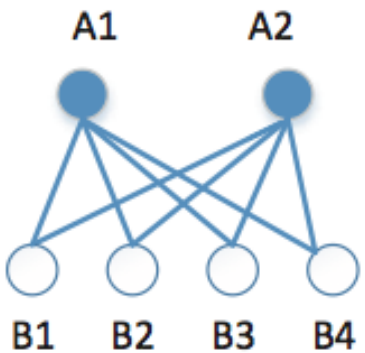
- The **first-order** proximity in a network is the **local** pairwise proximity between two vertices.
- For each pair of vertices linked by an edge (v_i, v_j), w_{ij} is the weight of the edge: indicates the first-order proximity between v_i and v_j.
- If no edge is observed between v_i and v_j, their **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: Large-scale 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 $f_G : V \rightarrow R^d$, where $d \ll |V|$. In the space R^d, both the **first-order** proximity and the **second-order** proximity between the vertices are preserved
- For each node $v_i \in V$, we use $\vec{u_i} \in R^d$ 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:

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

Combining first-order and second-order proximities

- A simple and effective way: train the LINE model which preserves the first-order proximity and second-order proximity separately and then **concatenate the embeddings trained by the two methods** together for each vertex.
- A more principled way to combine the two proximity is to jointly train the two objective function, **a future work?**.

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 $e_{i,j}$,

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

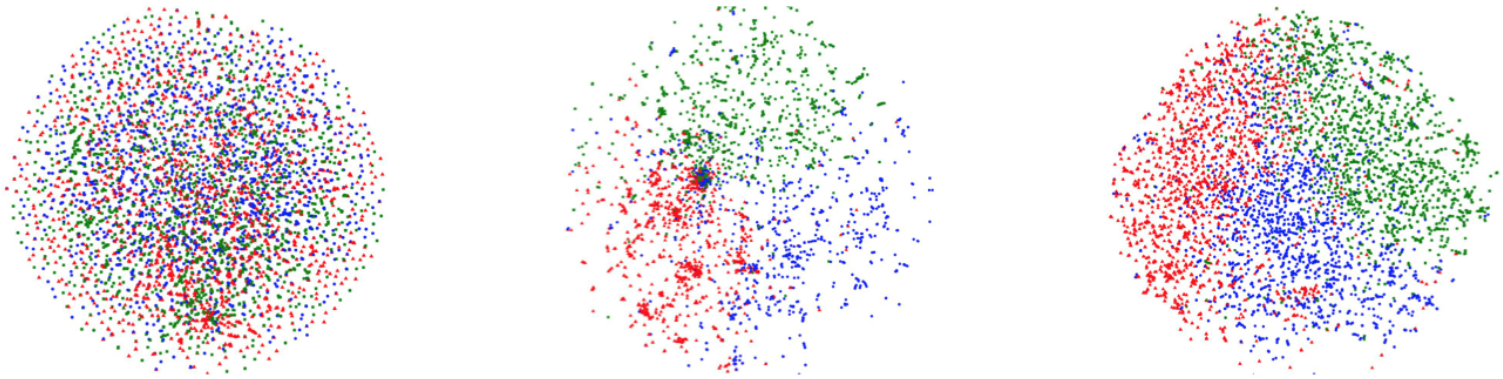
- We can use **asynchronous stochastic gradient algorithm (ASGD)** for optimizing above Eqn.

Performance

Table 8: Results of multi-label classification on DBLP(PaperCitation) network.

Metric	Algorithm	10%	20%	30%	40%	50%	60%	70%	80%	90%
Micro-F1	DeepWalk	52.83	53.80	54.24	54.75	55.07	55.13	55.48	55.42	55.90
	LINE(2nd)	58.42	59.58	60.29	60.78	60.94	61.20	61.39	61.39	61.79
		(60.10)**	**(61.06**)**	**(61.46**)**	**(61.73**)**	**(61.85**)**	**(62.10**)**	**(62.21**)**	**(62.25**)**	**(62.80**)**
Macro-F1	DeepWalk	43.74	44.85	45.34	45.85	46.20	46.25	46.51	46.36	46.73
	LINE(2nd)	48.74	50.10	50.84	51.31	51.61	51.77	51.94	51.89	52.16
		(50.22)**	**(51.41**)**	**(51.92**)**	**(52.20**)**	**(52.40**)**	**(52.59**)**	**(52.78**)**	**(52.70**)**	**(53.02**)**

Significantly outperforms DeepWalk at the: ** 0.01 and * 0.05 level, paired t-test.



(a) GF (b) DeepWalk (c) LINE(2nd)

Figure 2: Visualization of the co-author network. The authors are mapped to the 2-D space using the t-SNE package wi learned embeddings as input. Color of a node indicates the community of the author. Red: "data Mining," blue: "machi learning," green: "computer vision."

- Task: Predict which paper is published at the different conferences: AAAI, CIKM, ICML, KDD, NIPS, SIGIR, and WWW
- Result: The LINE(2nd) performs quite well and generates meaningful layout of the network (nodes with same colors are distributed closer). For more details, please check [5]

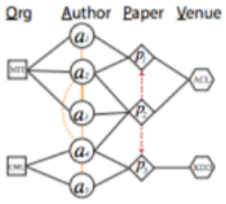
[5] Tang, J., Qu, M., Wang, M., Zhang, M., Yan, J. and Mei, Q., 2015, May. Line: Large-scale information network embedding. In WWW'15, pages 1067-1077.

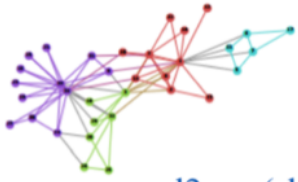
What are the fundamentals underlying the different models?

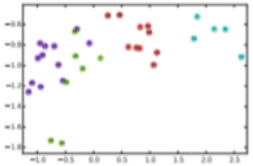
or

Can we unify the different network embedding approaches?

History of Network Embedding







2017 metapath2vec [Dong et al.]

2016 node2vec [Grover & Leskovec]

2015 LINE & PTE [Tang et al.]

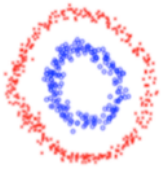
2014 DeepWalk [Perozzi et al.]



word2vec (skip-gram) [Mikolov et al.] 2013

2009 SocDim [Tang & Liu]

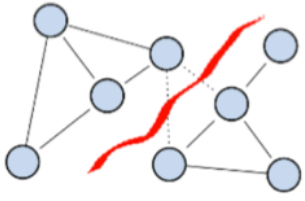
Spectral Clustering v.s. Kernel k-means [Dhillon et al.] 2005





Spectral Clustering [Ng et al.] 2002

Image Segmentation [Shi & Malik] 2000



1996 A large body of literature
[Pothen et al.] [Simon] [Bolla],
[Hagen & Kahng] [Hendrickson & Leland]
[Van Driessche & Roose], [Barnard et al.]
[Spielman & Teng], [Guattery & Miller]

1973 Fiedler Vector [Fiedler]

Spectral Partitioning [Donath, Hoffman]

Unifying DeepWalk, LINE, PTE, and node2vec into Matrix Forms[6]

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

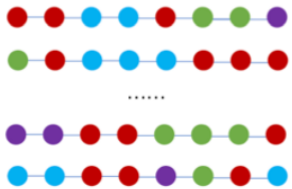
$\boldsymbol{A}$: $\boldsymbol{A}\in\mathbb{R}_+^{|V|\times|V|}$ is G's adjacency matrix with $\boldsymbol{A}_{i,j}$ as the edge weight between vertices i and j;
$\boldsymbol{D}_{\mathrm{col}}$: $\boldsymbol{D}_{\mathrm{col}}=\mathrm{diag}(\boldsymbol{A}^\top\boldsymbol{e})$ is the diagonal matrix with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

[6]Qiu J , Dong Y , Ma H , et al. Network Embedding as Matrix Factorization: Unifying DeepWalk, LINE, PTE, and node2vec[J]. 2017.

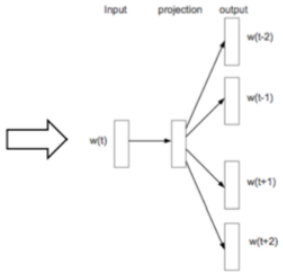
DeepWalk



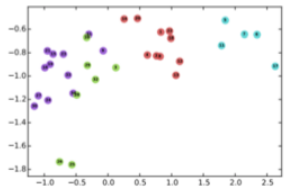
Input
G=(V,E)



Random
Walk



Skip-gram



Output:
Node
Embedding

Levy & Goldberg (NIPS 14)

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

b Number of negative samples

$\#(w,c)$ Co-occurrence of w and c

$\#(w)$ Occurrence of word w

$|\mathcal{D}|$ Total number of word-context pairs

$\#(c)$ Occurrence of context c

Skip-gram with Negative Sampling

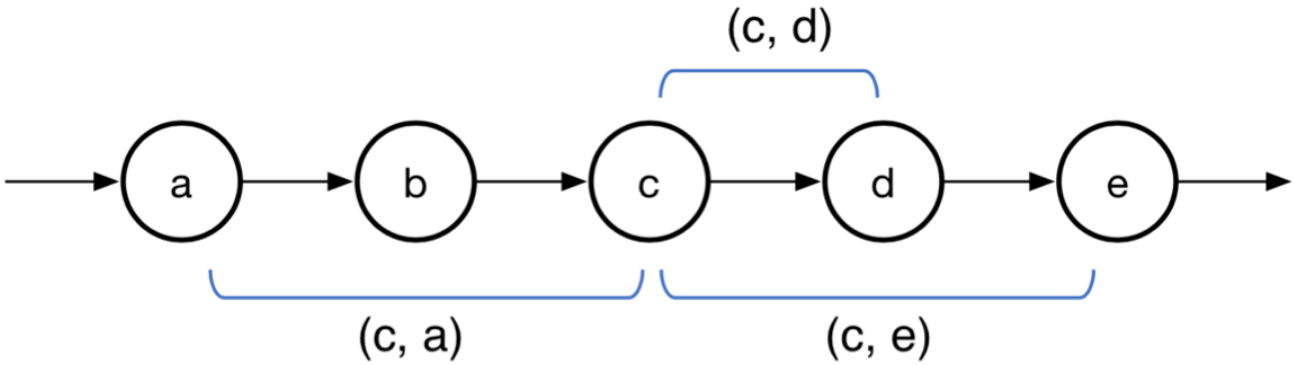
- SGNS maintains a multiset $\mathcal{D}$ which counts the occurrence of each word-context pair (w, c).
- Objective:

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

  where $\boldsymbol{x}_w, \boldsymbol{y}_c \in \mathbb{R}^d$, g is the sigmoid function, and b is the number of negative samples for SGNS.
- For sufficiently large dimensionality d, equivalent to factorizing PMI matrix (Levy & Goldberg, NIPS'14):

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

DeepWalk



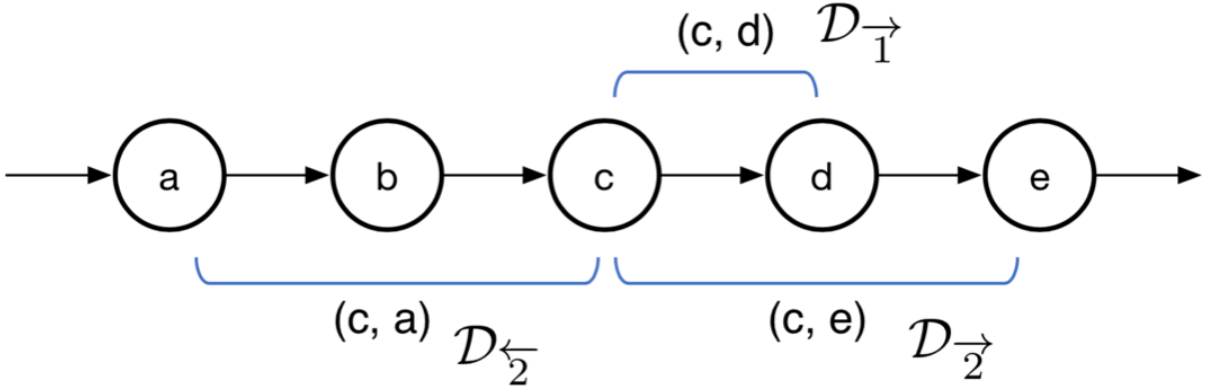
Question

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

DeepWalk

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

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



Some observations

- Observation 1:

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

- Observation 2:

$$\frac{\#(w,c)}{|\mathcal{D}|}=\frac{1}{2T}\sum_{r=1}^{T}\left(\frac{\#(w,c)_{\overrightarrow{r}}}{|\mathcal{D}_{\overrightarrow{r}}|}+\frac{\#(w,c)_{\overleftarrow{r}}}{|\mathcal{D}_{\overleftarrow{r}}|}\right).$$

Sufficient to characterize $\frac{\#(w,c)_{\overrightarrow{r}}}{|\mathcal{D}_{\overrightarrow{r}}|}$ and $\frac{\#(w,c)_{\overleftarrow{r}}}{|\mathcal{D}_{\overleftarrow{r}}|}$.

DeepWalk - Theorems

Theorem

Denote $\boldsymbol{P} = \boldsymbol{D}^{-1}\boldsymbol{A}$, when the length of random walk $L \to \infty$,

$$\frac{\#(w,c)_{\overrightarrow{r}}}{|\mathcal{D}_{\overrightarrow{r}}|} \xrightarrow{p} \frac{d_w}{\mathrm{vol}(G)}\left(\boldsymbol{P}^r\right)_{w,c} \text{ and } \frac{\#(w,c)_{\overleftarrow{r}}}{|\mathcal{D}_{\overleftarrow{r}}|} \xrightarrow{p} \frac{d_c}{\mathrm{vol}(G)}\left(\boldsymbol{P}^r\right)_{c,w}.$$

Theorem

When the length of random walk $L \to \infty$, we have

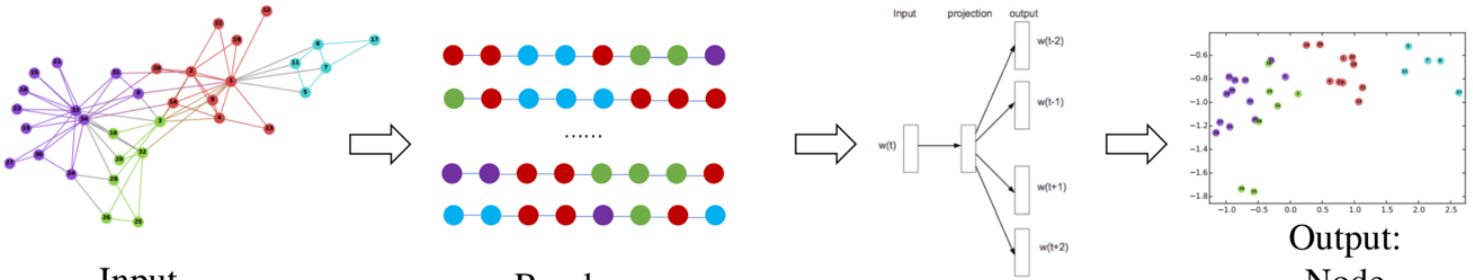
$$\frac{\#(w,c)}{|\mathcal{D}|} \xrightarrow{p} \frac{1}{2T}\sum_{r=1}^{T}\left(\frac{d_w}{\mathrm{vol}(G)}\left(\boldsymbol{P}^r\right)_{w,c} + \frac{d_c}{\mathrm{vol}(G)}\left(\boldsymbol{P}^r\right)_{c,w}\right).$$

Theorem

For DeepWalk, when the length of random walk $L \to \infty$,

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

DeepWalk - Conclusion



Input G=(V,E) | Random Walk | Skip-gram | Output: Node Embedding

Levy & Goldberg (NIPS 14)

$$\log\left(\frac{\mathrm{vol}(G)}{b}\left(\frac{1}{T}\sum_{r=1}^{T}\left(\boldsymbol{D}^{-1}\boldsymbol{A}\right)^{r}\right)\boldsymbol{D}^{-1}\right) \Leftarrow \log\left(\frac{\#(w,c)\,|\mathcal{D}|}{b\#(w)\#(c)}\right)$$

$\boldsymbol{A}$ Adjacency matrix $\quad \mathrm{vol}(G) = \sum_i \sum_j \boldsymbol{A}_{i,j}$

$\boldsymbol{D}$ Degree matrix $\quad b$ Number of negative samples

LINE

- Objective of LINE:

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

- Align it with the Objective of SGNS:

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

- LINE is actually factorizing

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

- Recall DeepWalk's matrix form:

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

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

node2vec - 2nd Order Random Walk

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

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

Stationary Distribution

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

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

node2vec as Implicit Matrix Factorization

Theorem

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

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

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

NetMF

- Factorize the DeepWalk matrix:

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

- For numerical reason, we use truncated logarithm — $\widetilde{\log}(x) = \log\left(\max(1, x)\right)$

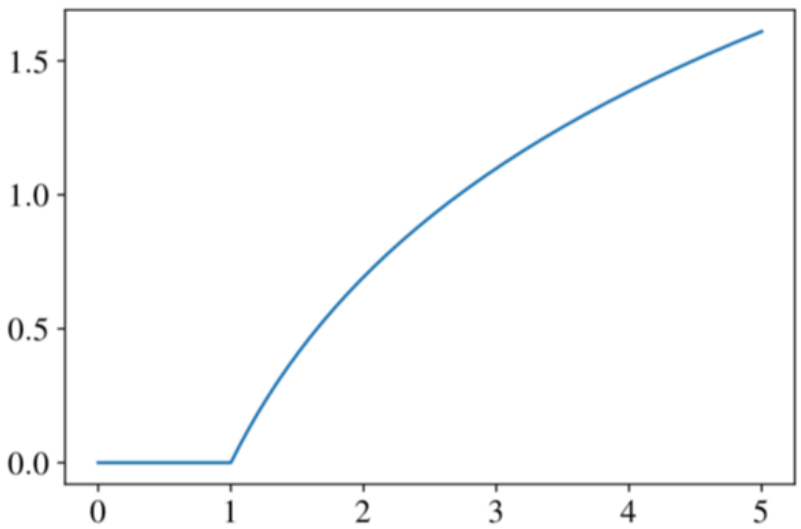


Figure 3: Truncated Logarithm

NetMF - Results

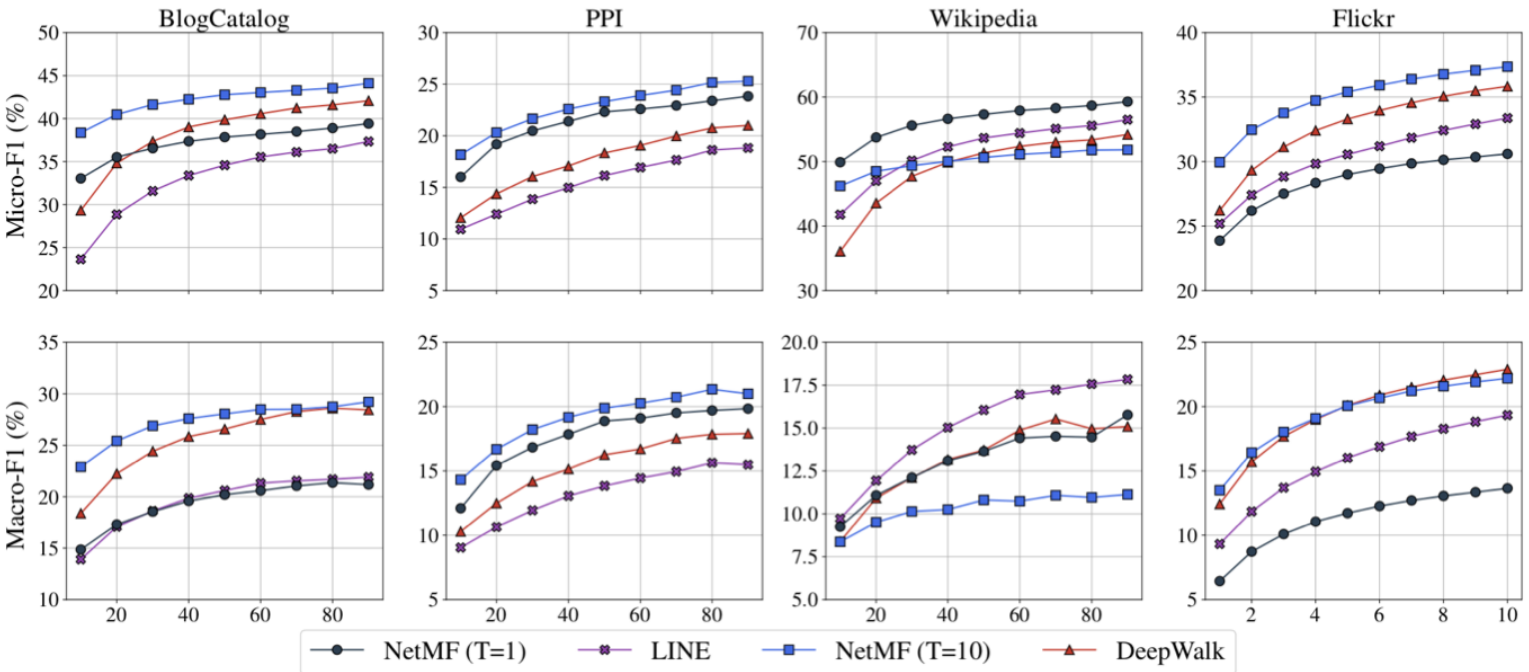
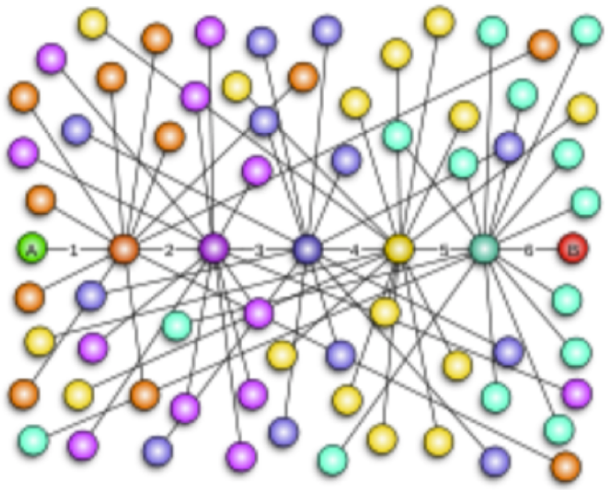
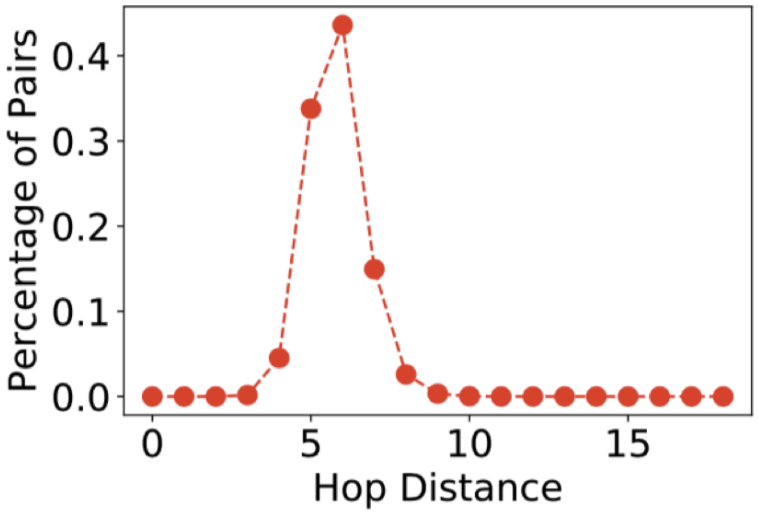


Figure 5: Predictive performance on varying the ratio of training data. The x-axis represents the ratio of labeled data (%), and the y-axis in the top and bottom rows denote the Micro-F1 and Macro-F1 scores respectively.

Challenge in NetMF



Small world



Academic graph

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

Random-walk Matrix Polynomial Sparsification

Definition

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

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

Theorem

[CCL+15] 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, in time* $O(T^2 m \epsilon^{-2} \log^2 n)$, *a* $(1+\epsilon)$*-spectral sparsifier,* $\widetilde{\boldsymbol{L}}$, *with* $O(n \log n \epsilon^{-2})$ *non-zeros. For unweighted graphs, the time complexity can be reduced to* $O(T^2 m \epsilon^{-2} \log n)$.

NetSMF: Large-Scale Network Embedding

Setting $\alpha_1 = \cdots = \alpha_T = \frac{1}{T}$ in $\boldsymbol{L} = \boldsymbol{D} - \sum_{r=1}^{T} \alpha_r \boldsymbol{D} \left(\boldsymbol{D}^{-1}\boldsymbol{A}\right)^r$, we can observe the tight connection between random walk matrix polynomial and NetMF:

$$\begin{aligned}
&\log^\circ \left(\frac{\mathrm{vol}(G)}{b} \left(\frac{1}{T} \sum_{r=1}^{T} \left(\boldsymbol{D}^{-1}\boldsymbol{A}\right)^r \right) \boldsymbol{D}^{-1} \right) \\
=&\log^\circ \left(\frac{\mathrm{vol}(G)}{b} \boldsymbol{D}^{-1}(\boldsymbol{D} - \boldsymbol{L})\boldsymbol{D}^{-1} \right) \\
\approx&\log^\circ \left(\frac{\mathrm{vol}(G)}{b} \boldsymbol{D}^{-1}(\boldsymbol{D} - \widetilde{\boldsymbol{L}})\boldsymbol{D}^{-1} \right)
\end{aligned}$$

Idea

Rewrite the NetMF matrix in terms of random walk matrix polynomial $\boldsymbol{L}$, and further approximate $\boldsymbol{L}$ with $\widetilde{\boldsymbol{L}}$.

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

NetSMF Algorithm

The proposed NetSMF algorithm consists of three steps:

- Construct a random walk matrix polynomial sparsifier, $\widetilde{\boldsymbol{L}}$, by calling algorithm proposed in [CCL$^+$15].
- Construct a NetMF matrix sparsifier.

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

- Truncated singular value decomposition.

NetSMF Algorithm

Denote $\boldsymbol{M} = \boldsymbol{D}^{-1}\left(\boldsymbol{D} - \boldsymbol{L}\right)\boldsymbol{D}^{-1}$ in

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

and $\widetilde{\boldsymbol{M}}$ to be its sparsifier the we constructed.

Theorem

The singular value of $\widetilde{\boldsymbol{M}} - \boldsymbol{M}$ satisfies

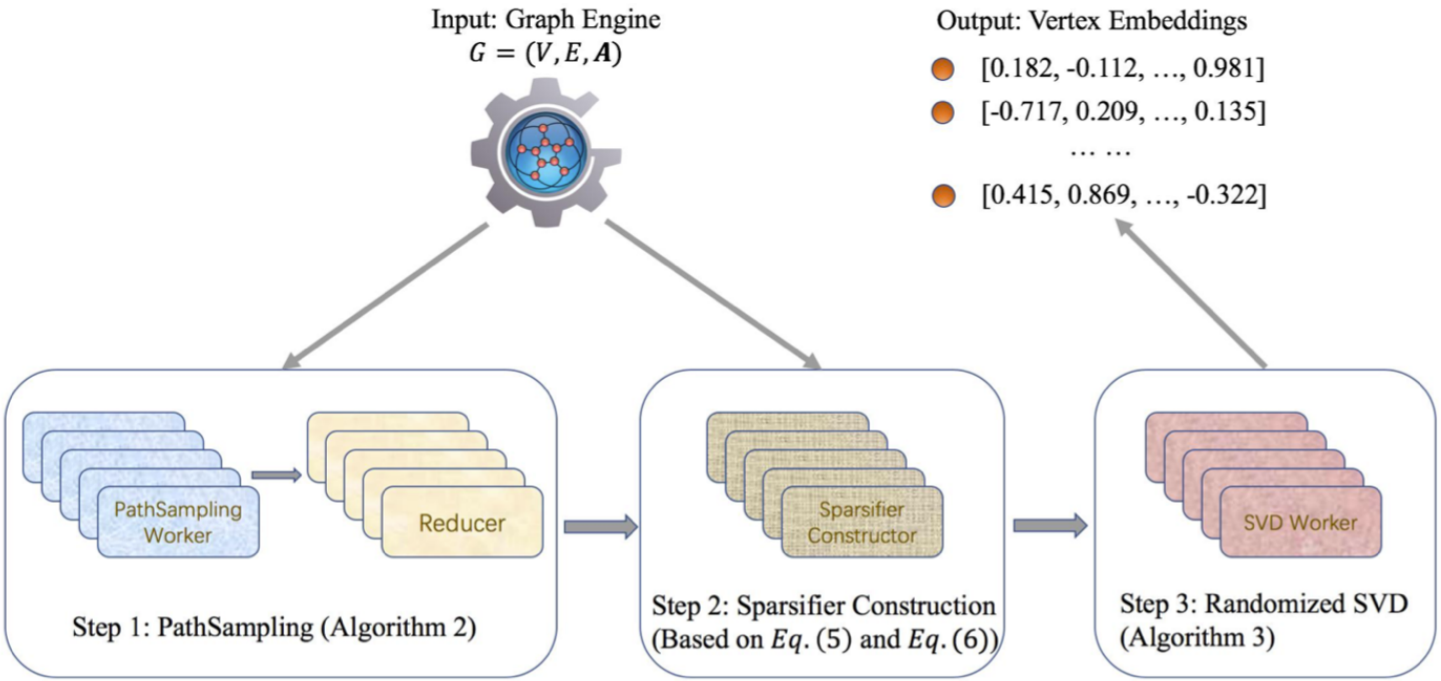
$$\sigma_i(\widetilde{\boldsymbol{M}} - \boldsymbol{M}) \leq \frac{4\epsilon}{\sqrt{d_i d_{\min}}}, \forall i \in [n].$$

Theorem

Let $\|\cdot\|_F$ be the matrix Frobenius norm. Then

$$\left\|\text{trunc_log}^{\circ}\left(\frac{\text{vol}(G)}{b}\widetilde{\boldsymbol{M}}\right) - \text{trunc_log}^{\circ}\left(\frac{\text{vol}(G)}{b}\boldsymbol{M}\right)\right\|_F \leq \frac{4\epsilon\,\text{vol}(G)}{b\sqrt{d_{\min}}}\sqrt{\sum_{i=1}^{n}\frac{1}{d_i}}.$$

NetSMF Algorithm



NetSMF Algorithm

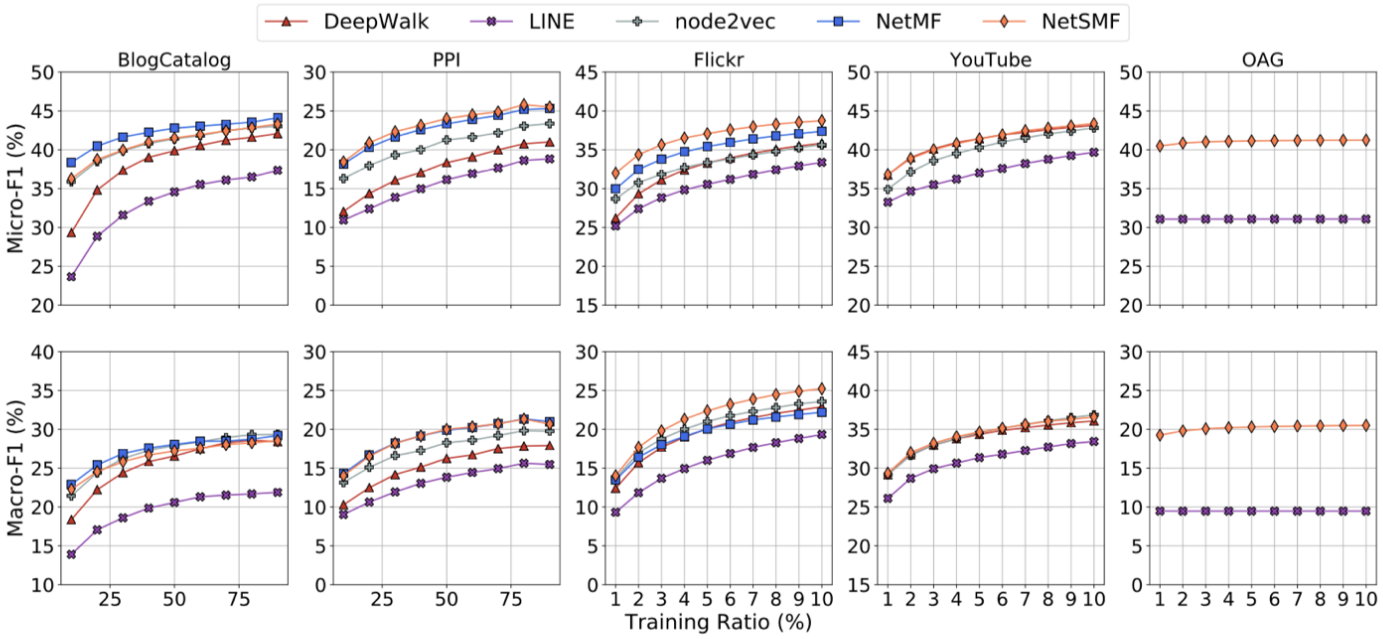
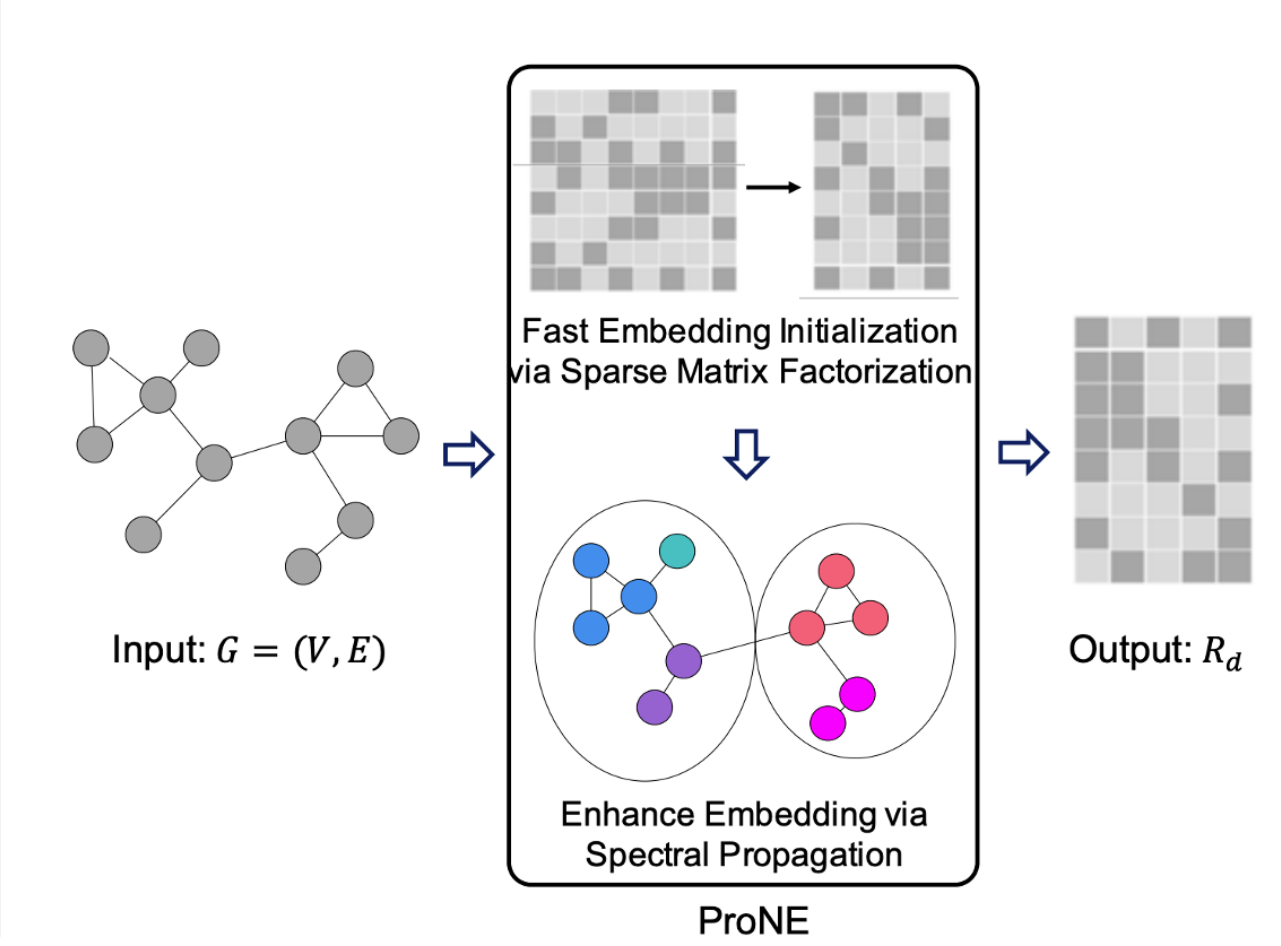


Figure 7: Predictive performance on varying the ratio of training data. The x-axis represents the ratio of labeled data (%), and the y-axis in the top and bottom rows denote the Micro-F1 and Macro-F1 scores respectively.

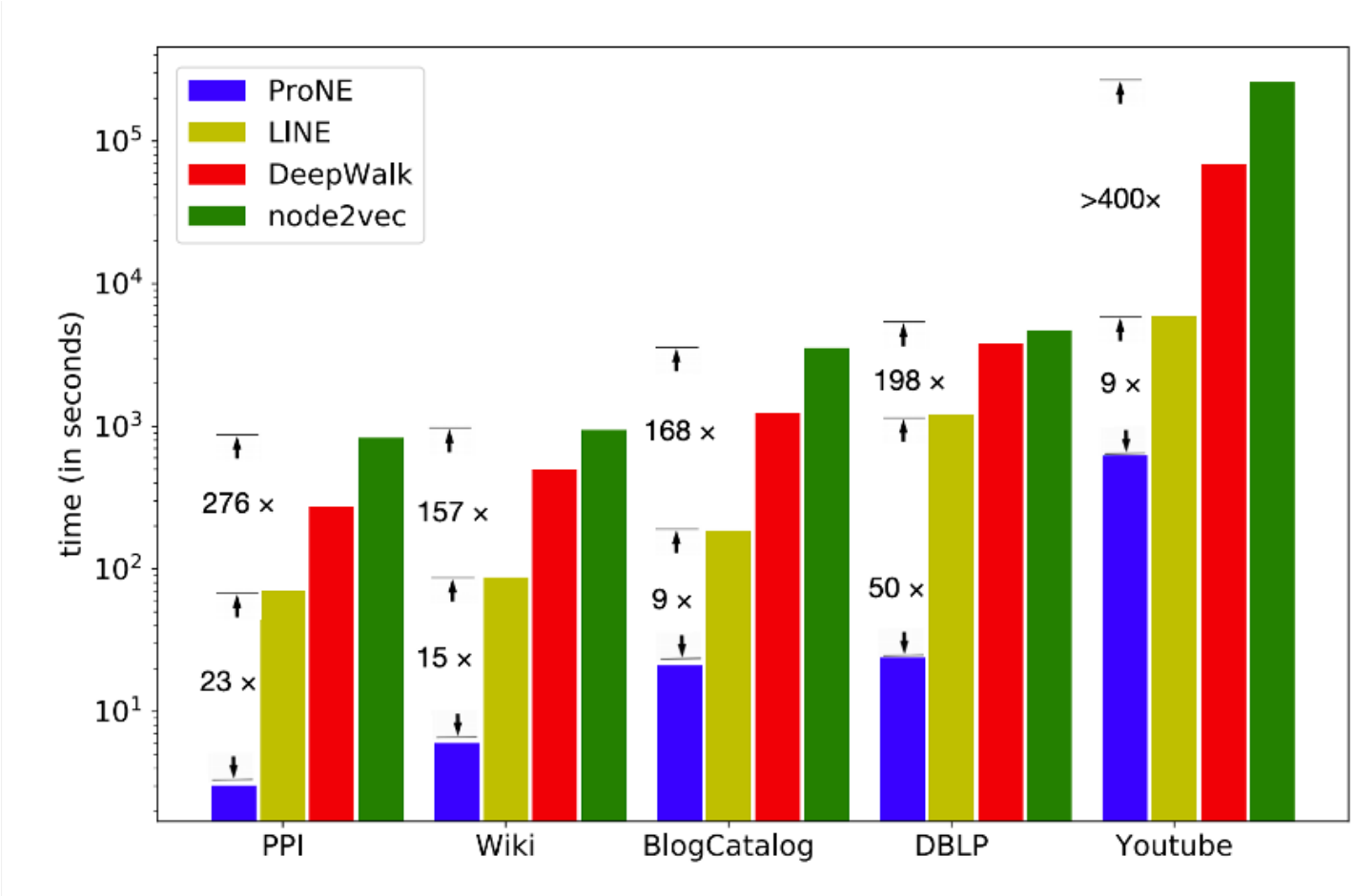
ProNE: Fast and Scalable Network Embedding



J. Zhang, Y. Dong, Y. Wang, J. Tang, and M. Ding. ProNE: Fast and Scalable Network Representation Learning. IJCAI'19.

ProNE Efficiency

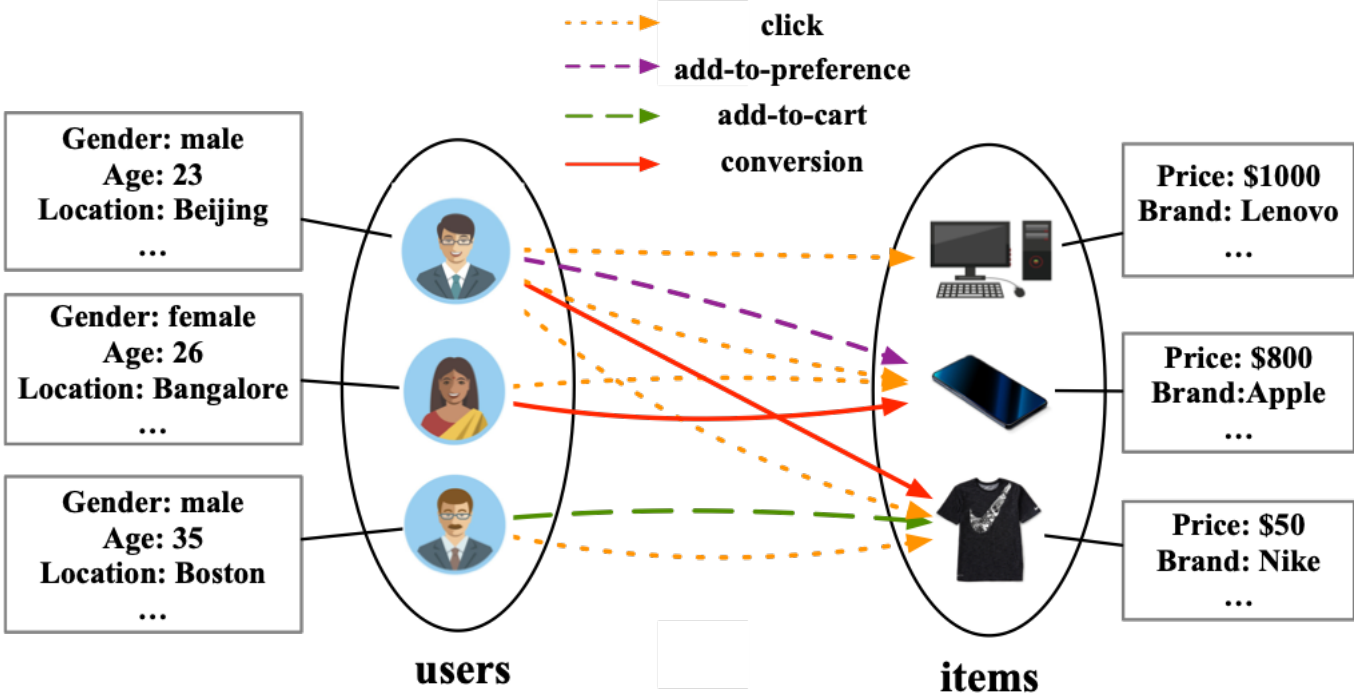
- ProNE offers 10-400X speedups (1 thread vs 20 threads)



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

Heterogeneous Network Embedding

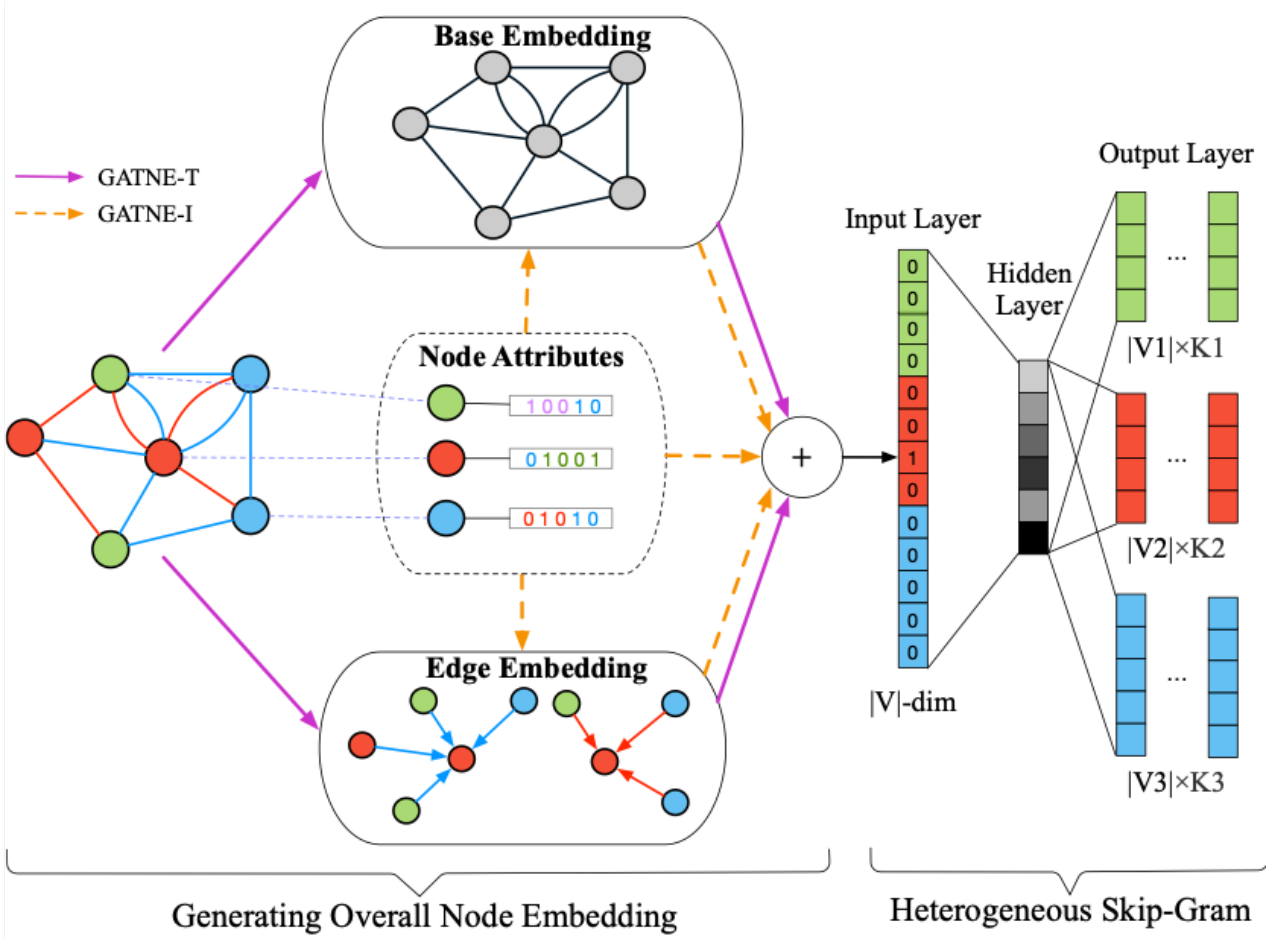
- Real-world network-structured applications are complicated, comprising not only multi-typed nodes and/or edges but also a rich set of attributes.



Summary of Network Embedding Methods

Network Type	Method	Heterogeneity		Attribute
		Node Type	Edge Type	
Homogeneous Network (HON)	DeepWalk [27] LINE [34] node2vec [10] NetMF [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 [33] metapath2vec [7] HERec [30]	Multi	Single	/
Attributed HEN (AHEN)	HNE [3]	Multi	Single	Attributed
Multiplex Heterogeneous Network (MHEN)	PMNE [22] MVE [29] MNE [43] mvn2vec [31]	Single	Multi	/
	GATNE-T	Multi	Multi	
Attributed MHEN (AMHEN)	*GATNE*-I	Multi	Multi	Attributed

GATNE: Multiplex Heterogeneous Network Embedding



Y. Cen, X. Zou, J. Zhang, H. Yang, J. Zhou and J. Tang. Representation Learning for Attributed Multiplex Heterogeneous Network. KDD'19.
Code available at `https://github.com/THUDM/GATNE`.

Contents

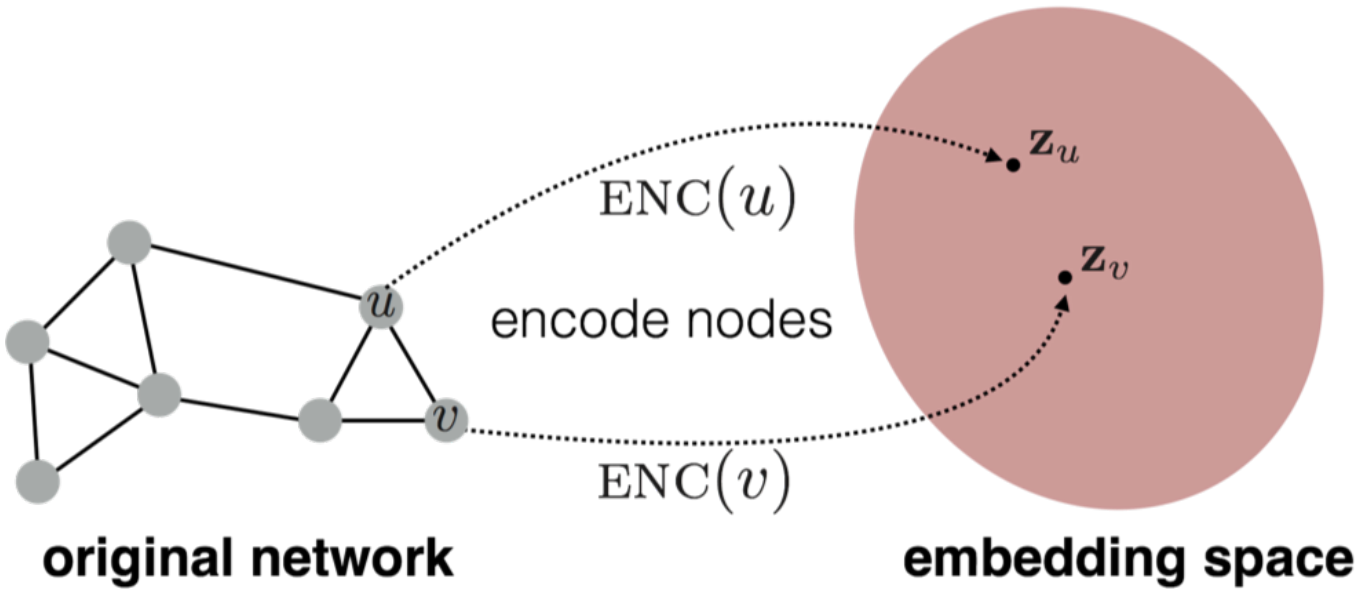
1 Introduction

2 Node Embeddings

3 Graph Neural Networks

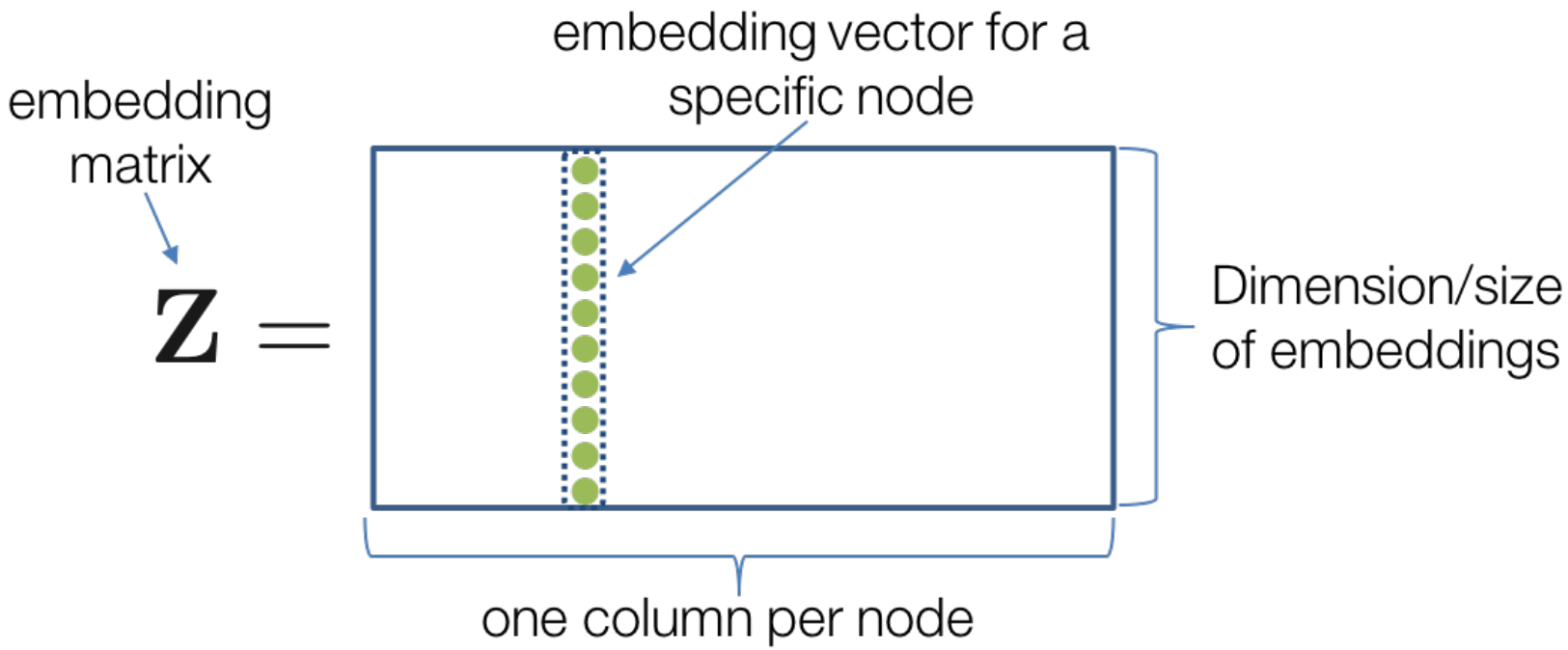
Embedding Nodes

- The goal is to encode nodes so that similarity in the embedding space (e.g., dot product) approximates similarity in the original network.



From Shallow to Deep

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

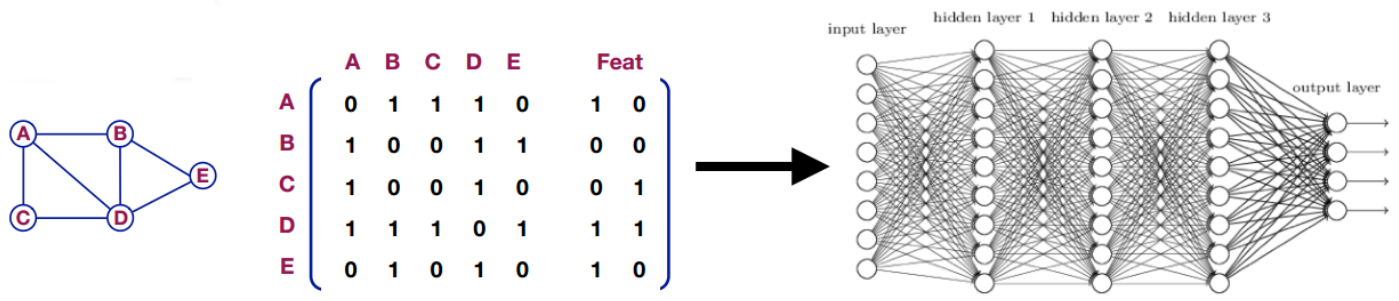


From Shallow to Deep

- 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

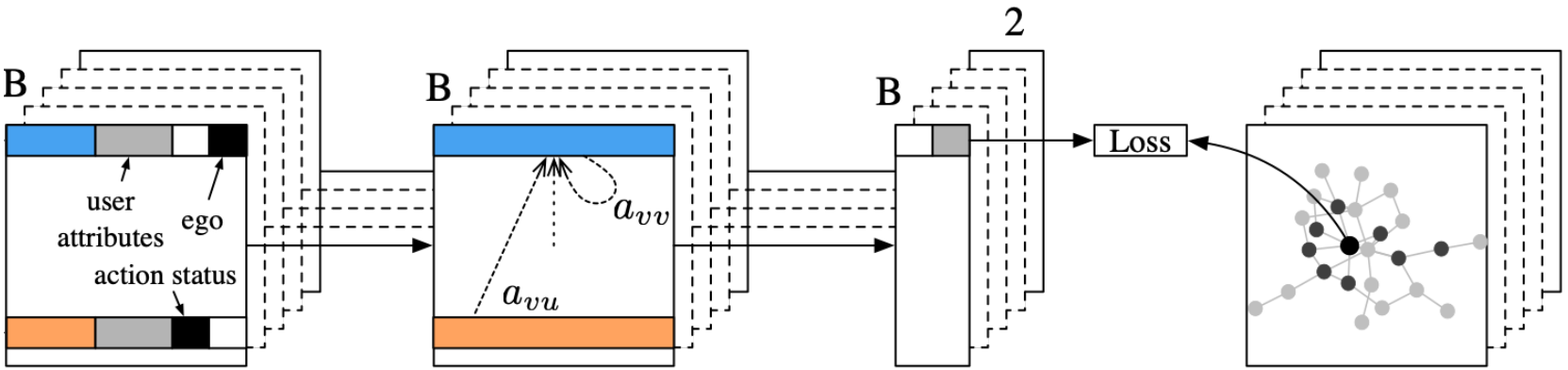
- Taking adjacency matrix A and feature matrix X into deep(fully connected neural network).



- 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 can be viewed as multi-layer graph convolution network

- 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.



GCN Model Architecture[7]

- GCN: Semi-Supervised Classification with Graph Convolutional Networks.
  - We consider spectral convolutions on graphs defined as the multiplication of a signal $x \in \mathrm{R}^N$ with a filter g_θ parameterized by $\theta \in \mathrm{R}^N$ in the Fourier domain
  - With several approximations like the Chebyshev polynomials, we can get the following equation

$$g'_\theta * x \approx \theta'_0 x + \theta'_1 D^{-\frac{1}{2}} A D^{-\frac{1}{2}} x$$

  - Set $\theta = \theta'_0 = -\theta'_1$ (can reduce overfitting)

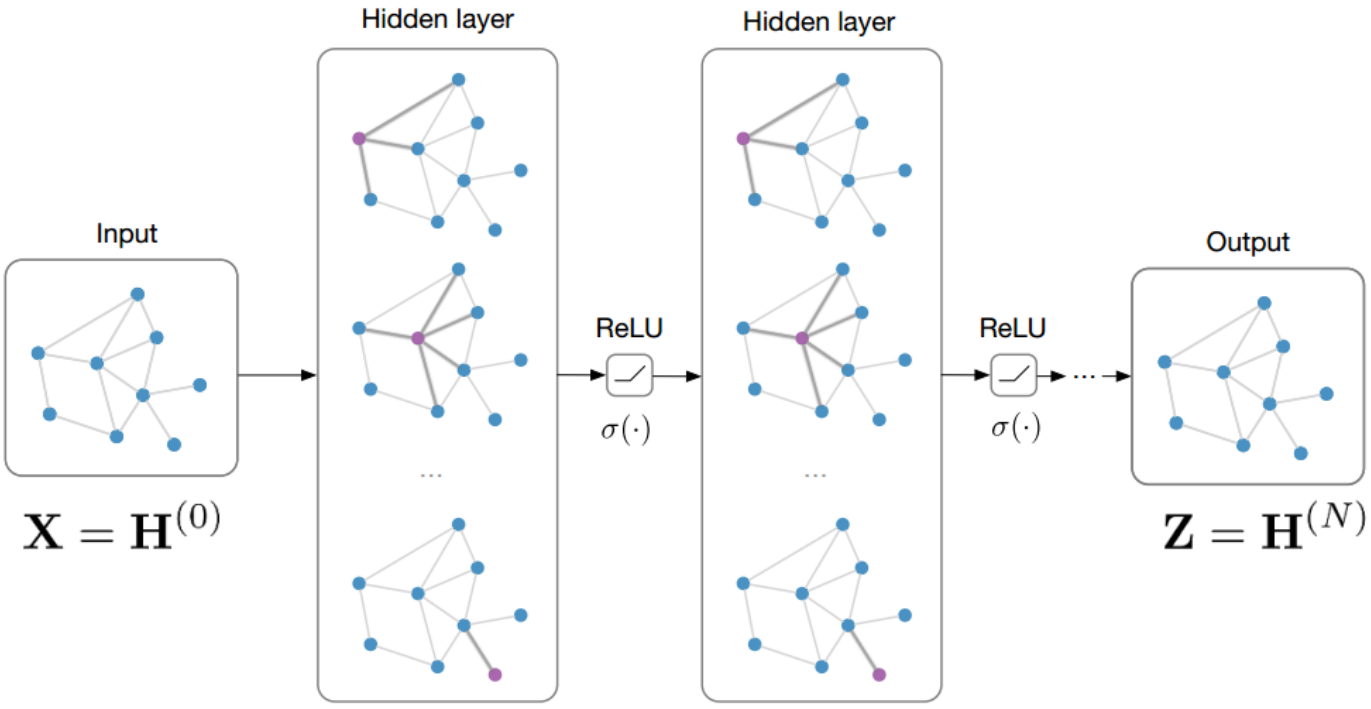
$$g_\theta * x \approx \theta \left(I_N + D^{-\frac{1}{2}} A D^{-\frac{1}{2}} \right) x$$

  - to alleviate gradients exploding/vanishing problems of $I_N + D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$, use **renormalizatin trick**: $I_N + D^{-\frac{1}{2}} A D^{-\frac{1}{2}} \rightarrow \hat{\mathbf{A}} \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}}$

[7] Kipf, T.N. and Welling, M., 2016. Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907.

GCN Model Architecture

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



- 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 $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$

GCN :Semi-supervised classification

- Traditional **embedding-based approaches**: DeepWalk, LINE
  Two-step piplines:
  - Get Embedding for every node
  - Train classifier on node embedding
- **GCN Idea**: Train graph-based classifier end-to-end using GCN, evaluate loss $\mathcal{L}$ on labeled nodes:

$$\mathcal{L} = -\sum_{l \in \mathcal{Y}_L} \sum_{f=1}^{F} Y_{lf} ln Z_{lf}$$

  where $\mathcal{Y}_L$ is the set of labeled node indices that have labels, Y is the label matrix and Z is GCN output(after softmax).

GCN : Performance

Model: 2-layer GCN $Z = f(X, A) = \mathrm{softmax}\Big(\hat{A}\ \mathrm{ReLU}\Big(\hat{A}XW^{(0)}\Big)\ W^{(1)}\Big)$

Dataset statistics

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

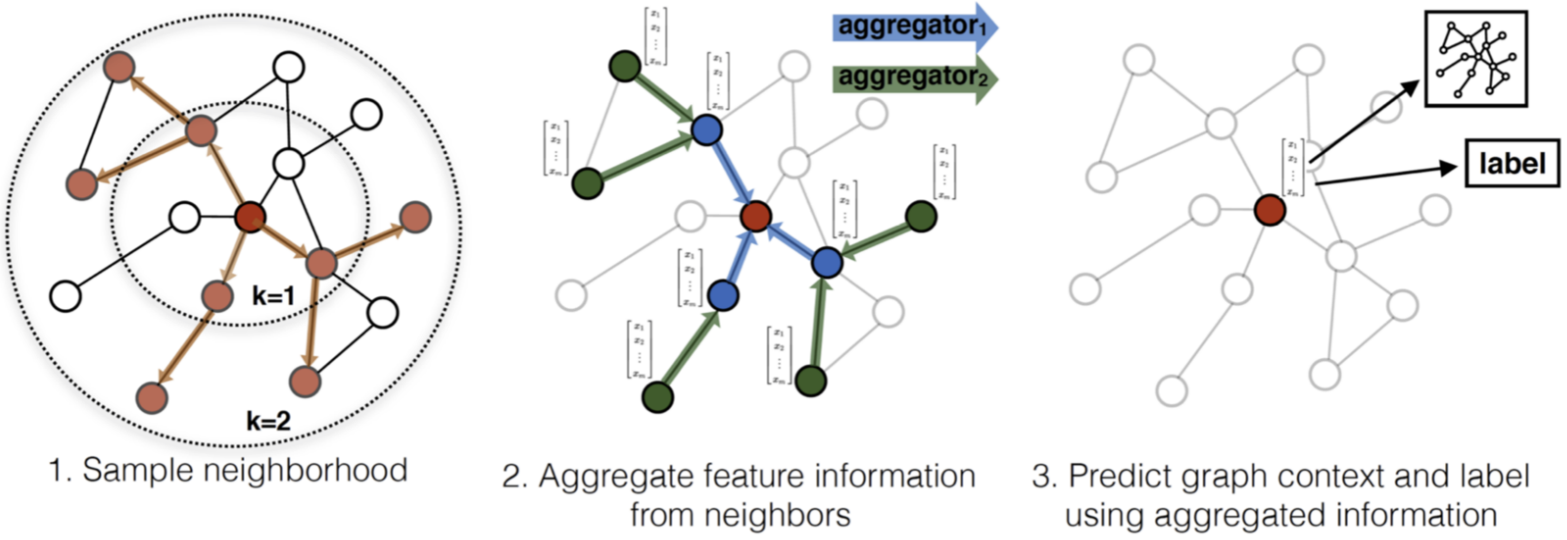
Classification results (accuracy)

no input features

Method	Citeseer	Cora	Pubmed	NELL
ManiReg [3]	60.1	59.5	70.7	21.8
SemiEmb [24]	59.6	59.0	71.1	26.7
LP [27]	45.3	68.0	63.0	26.5
DeepWalk [18]	43.2	67.2	65.3	58.1
Planetoid* [25]	64.7 (26s)	75.7 (13s)	77.2 (25s)	61.9 (185s)
GCN (this paper)	**70.3** (7s)	**81.5** (4s)	**79.0** (38s)	**66.0** (48s)
GCN (rand. splits)	67.9 ± 0.5	80.1 ± 0.5	78.9 ± 0.7	58.4 ± 1.7

GraphSAGE Model Arthitecture[8]

- 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

[8]Hamilton W, Ying Z, Leskovec J. Inductive representation learning on large graphs[C]//Advances in Neural Information Processing Systems. 2017: 1024-1034. MLA

GraphSAGE vs GCN

- GCN Neighborhood Aggregation:

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

- GraphSAGE Neighborhood Aggregation:

$$h_v^k = \sigma([W_k \odot AGG(\{h_u^{k-1}, \forall u \in N_s(v)\}), B_k h_v^{k-1}])$$

  where $N_s(v) \subseteq N(v)$ is a fixed-size set of neighbors sampled from full neighborhood sets $N(v)$. AGG is generalized aggregation function.

GraphSAGE Variants

- Mean:

$$AGG = \sum_{u \in N(v)} \frac{h_u^{k-1}}{|N(v)|}$$

- Pool:
  - Transform neighbor vectors and apply symmetric vector function.

$$AGG = \gamma(\{Qh_u^{k-1}, \forall u \in N(v)\})$$

  where γ is element-wise mean/max
- LSTM
  - Apply LSTM to random permutation of neighbors.

$$AGG = LSTM([h_u^{k-1}, \forall u \in \pi(N(v))])$$

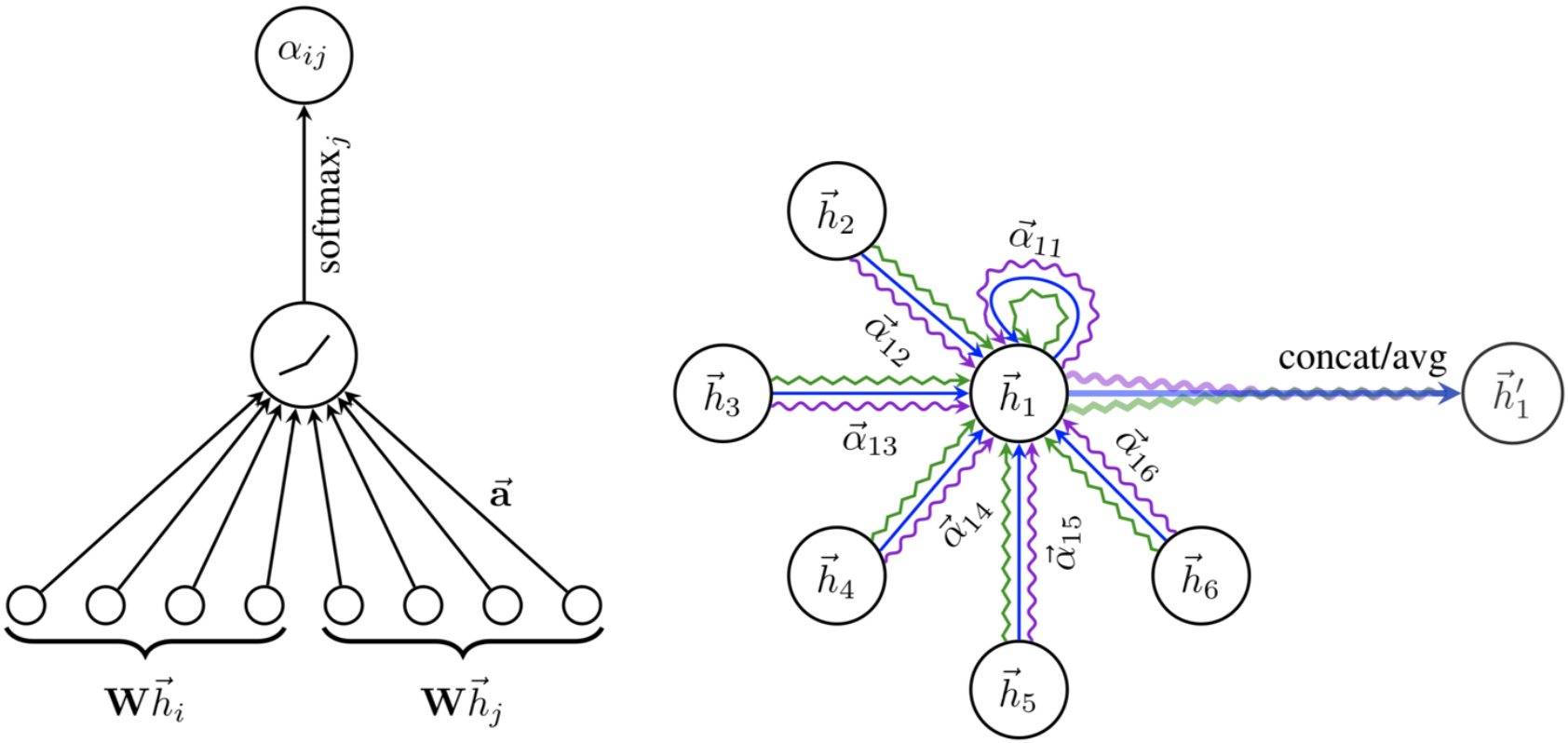
GraphSAGE Results

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(GATs)[9]

- Applying self-attention into graph data.



[9]Veličković P, Cucurull G, Casanova A, et al. Graph attention networks[J]. arXiv preprint arXiv:1710.10903, 2017. MLA

GAT vs GCN

- GCN Neighborhood Aggregation:

$$\vec{h}_i' = \sigma\left(W \sum_{j \in N(i) \bigcup i} \frac{\vec{h}_j}{\sqrt{|N(i)||N(j)|}}\right)$$

- GAT Neighborhood Aggregation:

$$\vec{h}_i' = \sigma(\frac{1}{K} \sum_{k=1}^{K} \sum_{j \in N(i) \bigcup i} \alpha_{ij}^k W^k \vec{h}_j)$$

where α_{ij} is attention coefficient between node i and node j:

$$\alpha_{ij} = \frac{\exp(\text{LeakyReLU}(\vec{a}^T [W\vec{h}_i || W\vec{h}_j]))}{\sum_{k \in N(i) \bigcup i} \exp(\text{LeakyReLU}(\vec{a}^T [W\vec{h}_i || W\vec{h}_k]))}$$

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

Extensions

- Node Ranking-Aware Rescaling
- Network Sampling based GCN
- Tackling Oversmoothing in GNNs

NRGCN: Node Ranking-aware GCN

- Intuition: Complex propagation or diffusion processes over a network are driven by the diversity of nodes' status and influence— as measured by their relative **rankings** in the network.
- NRGCN incorporates the node ranking into neighborhood aggregation step in GCN.

NRGCN: Node Ranking-aware GCN

- Neighborhood aggregation in GCN:

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

- Neighborhood aggregation in NRGCN (for node *i*):

$$H_i^{(l+1)} = \sigma_i^{(l)}[(P^{(l)}\hat{A}Q^{(l)}H^{(l)}W^{(l)})_i]$$

  - *P* and *Q*: diagonal matrix, node ranking-aware rescaling
  - σ_i : node ranking-aware encoder
  - P_{ii}, Q_{ii}, and σ_i are adaptively adjusted according to the network ranking of node i

NRGCN: Node Ranking-aware GCN

- Network state $\vec{s}_1$
- Current node ranking:

$$r_i = \vec{s}_1^T (HW)_i$$

- Node ranking-aware rescaling:

$$Q_{ii} = \text{sigmoid}[r_i] \qquad P_{ii} = \frac{1}{\sum_j \hat{A}_{ij} Q_{jj}}$$

- Current node ranking:

$$r_i = \vec{s}_2^T (P\hat{A}QHW)_i$$

- Node ranking-aware encoder:

$$\sigma_i(x) = \begin{cases} x & , x > 0 \\ \text{sigmoid}(r_i)(e^x - 1) & , x \leq 0 \end{cases}$$

NRGCN: Node Ranking-aware GCN

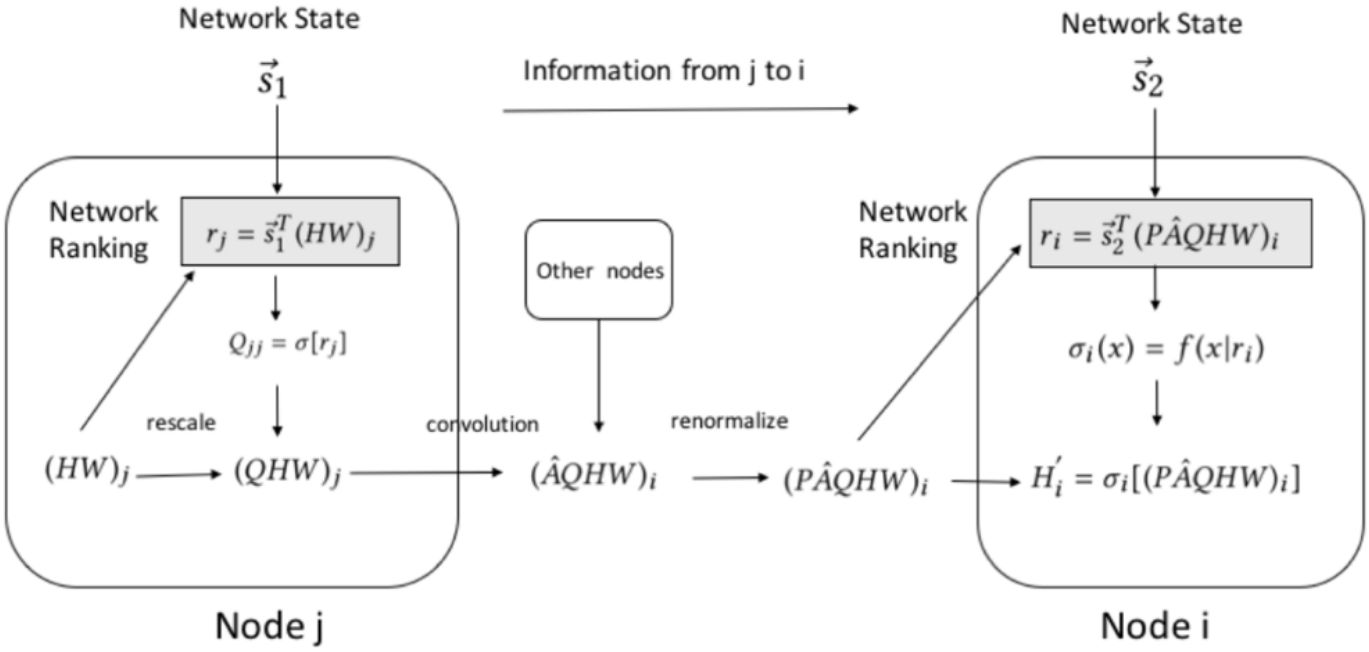


Figure 1: The NRGCN model. The signal propagation from source j to the target neighbor i involves j's rescaling function according to its node ranking, i's convolutional operation which sums over the signals in its neighborhood and again rescales (renormalizes) the resulting signal, and finally i choosing an encoder according to its current network ranking.

NRGCN: Model Enhancements

- NRGCN layer:

$$H_i^{'} = \sigma_i[(P\hat{A}QHW)_i]$$

- Multi-head Propagation mechanism:

$$H_i^{'} = \Big\Vert_{k=1}^{K} \sigma_i^{(k)}[(P^{(k)}\hat{A}Q^{(k)}HW^{(k)})_i]$$

- Multi-hop variants:

$$H_i^{'} = \sigma_i[(P\hat{A}^k QHW)_i]$$

Generalization to Graph Attention

- When P normalizes signals in different range, the model can function as node, edge, and path attention mechanisms

(Node attention)

$$\begin{cases} S = \hat{A}Q \\ \vec{p}_i = P_{ii} = \dfrac{1}{\vec{q}^T \vec{1}} \end{cases}$$

(Edge attention)

$$\begin{cases} S = \hat{A}Q \\ \vec{p} = \dfrac{1}{S\vec{1}} \end{cases}$$

(K-hop edge attention)

$$\begin{cases} S = \hat{A}^k Q \\ \vec{p} = \dfrac{1}{S\vec{1}} \end{cases}$$

(Path attention)

$$\begin{cases} S = \hat{A}Q_k \hat{A}Q_{k-1} \ldots \hat{A}Q_1 \\ \vec{p} = \dfrac{1}{S\vec{1}} \end{cases}$$

GAT can be considered as a special case of NRGCN

NSGCN: Network Sampling GCN

- Observation: structural dependency and information redundancy in graph-structured data.
- Sampling can help to explore the network information!!!

$\text{NSGCN}_{\text{half}}$: a test model

- Sample half of the nodes as set C

$$\begin{cases} Pr(X_i^{'} = X_i) = 0.5, & i \in C. \\ Pr(X_i^{'} = \vec{0}) = 0.5, & i \notin C. \end{cases}$$

- Recover the feature matrix by propagation

$$\widetilde{X} = \frac{1}{k} \sum_{i}^{k} \hat{A}^i X^{'}$$

- $\text{NSGCN}_{\text{half}}$ (GCN model with partial nodes' features) can generate very close performance to the ordinary GCN.

NSGCN_{dp}: Exponential ensemble

- Different sampled subnetworks may provide different views of the original network.
- Dropout-like ensemble model NSGCN_{dp}:
  - Exponential ensemble of $\text{NSGCN}_{\text{half}}$
  - Sample half of the nodes and do the $\text{NSGCN}_{\text{half}}$ training process at each training epoch
  - Inference with the entire feature matrix and a discount factor 0.5
- Training Loss

$$Loss_{c1} = -\sum_{i \in V^L} \sum_{l}^{|\mathcal{Y}|} Y_{i,l} \ln Z_{i,l}$$

- Inference

$$\hat{Z} = GCN\left(\frac{0.5}{k} \sum_{i}^{k} \hat{A}^i X\right)$$

NSGCN_{dm}: Disagreement Minimization

- Minimize the disagreement of two complementary NSGCN_{dp}s (on C and V-C at each training epoch)
- The objective to minimize the disagreement at a certain epoch can be obtained by Jensen-Shannon-divergence:

$$Loss_{JS} = \frac{1}{2} \sum_{i \in V} \sum_{l}^{|\mathcal{Y}|} \left(Z'_{i,l} \ln \frac{Z'_{i,l}}{Z''_{i,l}} + Z''_{i,l} \ln \frac{Z''_{i,l}}{Z'_{i,l}} \right)$$

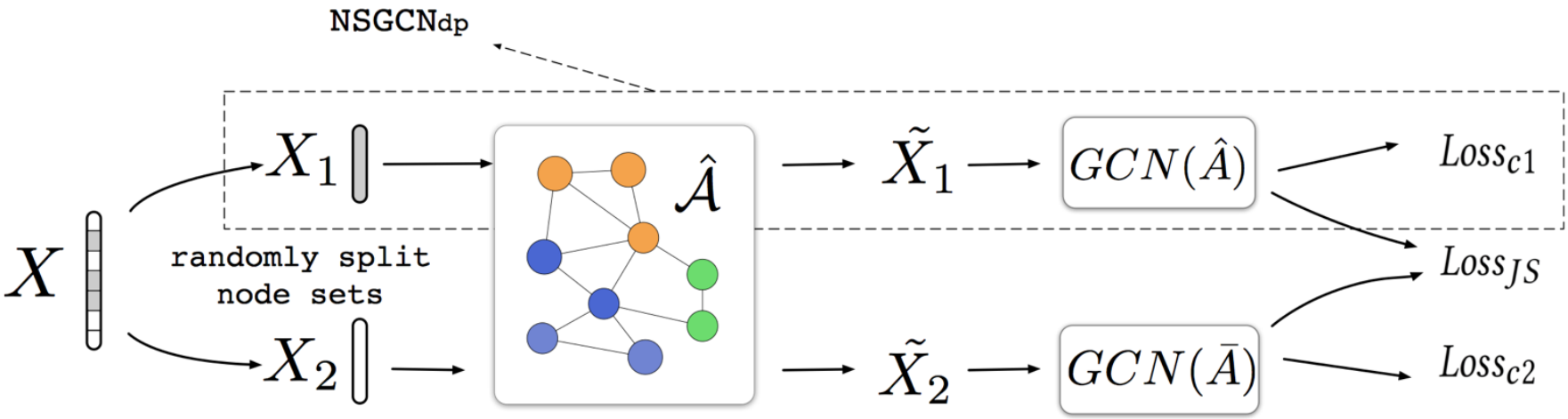
- Training Loss

$$Loss_{dm} = (Loss_{c1} + Loss_{c2})/2 + \lambda Loss_{JS}$$

- Inference

$$\hat{Z} = (\hat{Z}' + \hat{Z}'')/2$$

NSGCN: Network Sampling GCN



Experiments (datasets)

	Cora	Citeseer	Pubmed	PPI
Nodes	2,708	3,327	19,717	56,944
Edges	5,429	4,732	44,338	818,716
Classes	7	6	3	121
Features	1,433	3,703	500	50
Training Nodes	140	120	60	44,906
Validation Nodes	500	500	500	6,514
Test Nodes	1,000	1,000	1,000	5,524
Label Rate	0.052	0.036	0.003	0.789

Experiments (transductive)

Table 2: Summary of classification accuracy (%).

Category	Method	Cora	Citeseer	Pubmed
Non-Graph Convolution	MLP	55.1	46.5	71.4
	ManiReg	59.5	60.1	70.7
	SemiEmb	59.0	59.6	71.1
	LP	68.0	45.3	63.0
	DeepWalk	67.2	43.2	65.3
	ICA	75.1	69.1	73.9
	Planetoid	75.7	64.7	77.2
	GraphSGAN	83.0±1.3	73.1±1.8	–
Graph Convolution	Chebyshev	81.2	69.8	74.4
	GCN	81.5	70.3	79.0
	MoNet	81.7±0.5	–	78.8±0.3
	DPFCNN	83.3±0.5	72.6±0.8	–
	GAT	83.0±0.7	72.5±0.7	79.0±0.3
	GAT-16*	83.3±0.6	72.4±0.7	78.8±0.3
	NRGCN (ours)	**83.6±0.6**	**73.2±0.6**	**79.1±0.3**
	NRGCN (max)	85.8	74.6	79.7

Experiments (transductive)

Table 3: Summary of classification accuracy (graph sampling-based GCNs) (%).

Method	Cora	Citeseer	Pubmed
GCN	81.5	70.3	79.0
FastGCN	81.4±0.5	68.8±0.9	77.6±0.5
GraphSAGE (transductive)	78.9±0.8	67.4±0.7	77.8±0.6
NSGCN$_{half}$ (ours)	79.4±1.2	67.4±1.3	78.2±0.8
NSGCN$_{dp}$ (ours)	82.4±0.5	72.1±0.6	**79.4±0.4**
NSGCN$_{dm}$ (ours)	**84.5±0.5**	**72.7±0.4**	79.2±0.3

Experiments (inductive)

Table 4: Summary of classification accuracy (inductive learning) (%).

Method	PPI
Random	39.6
MLP	42.2
GraphSAGE-GCN	50.0
GraphSAGE-mean	59.8
GraphSAGE-LSTM	61.2
GraphSAGE-pool	60.0
GraphSAGE	76.8
GAT	97.3 ±0.2
NRGCN (ours)	**98.5 ±0.1**

PairNorm: Tackling Oversmoothing in GNNs

- Intuition: The performance of graph neural nets (GNNs) is known to gradually decrease with increasing number of layers. This decay is partly attributed to oversmoothing, where repeated graph convolutions eventually make node embeddings indistinguishable.
- PairNorm is a novel normalization layer based on a careful analysis of the graph convolution operator, which prevents all node embeddings from becoming too similar. [10]

[10] L. Zhao and L. Akoglu. PairNorm: Tackling Oversmoothing in GNNs. ICLR 2020.

PairNorm

- We start by establishing a connection between graph convolution and an optimization problem, that is graph-regularized least squares. Let $\overline{\mathbf{X}} \in \mathbb{R}^{n \times d}$ be a new node representation matrix, and then the problem is given as

$$\min_{\overline{\mathbf{X}}} \sum_{i \in \mathcal{V}} \|\overline{\mathbf{x}}_i - \mathbf{x}_i\|_{\tilde{\mathbf{D}}}^2 + \sum_{(i,j) \in \mathcal{E}} \|\overline{\mathbf{x}}_i - \overline{\mathbf{x}}_j\|_2^2 \tag{1}$$

  where $\|\mathbf{z}_i\|_{\tilde{\mathbf{D}}}^2 = \mathbf{z}_i^T \tilde{\mathbf{D}} \mathbf{z}_i$. The first term can be seen as total degree-weighted least squares. The second is a graph-regularization term that measures the variation of the new features over the graph structure.

- The goal of the optimization problem can be stated as estimating new "denoised" features $\overline{\mathbf{x}}_i$'s that are not too far off of the input features $\mathbf{x}_i$'s and are smooth over the graph structure.

PairNorm

- Let $\tilde{\mathbf{A}}_{\mathrm{sym}} = \tilde{\mathbf{D}}^{-1/2}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-1/2}$ and $\tilde{\mathbf{A}}_{\mathrm{rw}} = \tilde{\mathbf{D}}^{-1}\tilde{\mathbf{A}}$ denote symmetrically and nonsymmetrically normalized adjacency matrices with self-loops.
- The optimization problem has a closed form solution $\overline{\mathbf{X}} = \left(2\mathbf{I} - \tilde{\mathbf{A}}_{\mathrm{rw}}\right)^{-1}\mathbf{X}$, for which $\tilde{\mathbf{A}}_{\mathrm{rw}}\mathbf{X}$ is the first-order Taylor approximation, that is $\tilde{\mathbf{A}}_{\mathrm{rw}}\mathbf{X} \approx \overline{\mathbf{X}}$. By exchanging $\widetilde{\mathbf{A}}_{\mathrm{rw}}$ and $\widetilde{\mathbf{A}}_{\mathrm{rw}}$, we obtain the same form as the graph convolution, i.e., $\tilde{\mathbf{X}} = \tilde{\mathbf{A}}_{\mathrm{sym}}\mathbf{X} \approx \overline{\mathbf{X}}$. Therefore, graph convolution can be viewed as an approximate solution of the optimization problem, where it minimizes the variation over the graph structure while keeping the new representations close to the original.

PairNorm

- In order to avoid smoothing over nodes from different clusters, one solution is to add a negative term such as the sum of distances between disconnected pairs as follows.

$$\min_{\overline{\mathbf{x}}} \sum_{i \in \mathcal{V}} \|\overline{\mathbf{x}}_i - \mathbf{x}_i\|_{\tilde{\mathbf{D}}}^2 + \sum_{(i,j) \in \mathcal{E}} \|\overline{\mathbf{x}}_i - \overline{\mathbf{x}}_j\|_2^2 - \lambda \sum_{(i,j) \notin \mathcal{E}} \|\overline{\mathbf{x}}_i - \overline{\mathbf{x}}_j\|_2^2 \quad (2)$$

  where λ is a balancing scalar. By deriving the closed-form and approximating it with first-order Taylor expansion, one can get a revised graph convolution operator with hyperparameter λ.
- In this paper, the authors take a different route. Instead of a completely new graph convolution operator, they propose a general and efficient "patch" called PariNorm, that can be applied to any form of graph convolution having the potential of oversmoothing.

PairNorm

- Let $\tilde{\mathbf{X}}$ (the output of graph convolution) and $\dot{\mathbf{X}}$ respectively be the input and output of PairNorm. Observing that the output of graph convolution $\tilde{\mathbf{X}} = \tilde{\mathbf{A}}_{\text{sym}}\mathbf{X}$ only achieves the first goal, PairNorm serves as a normalization layer that works on $\tilde{\mathbf{X}}$ to achieve the second goal of keeping disconnected pair representations farther off. Specifically, PairNorm normalizes $\tilde{\mathbf{X}}$ such that the total pairwise squared distance $\text{TPSD}(\dot{\mathbf{X}}) := \sum_{i,j\in[n]} \|\dot{\mathbf{x}}_i - \dot{\mathbf{x}}_j\|_2^2$ is the same as $TPSD(\mathbf{X})$. That is,

$$\sum_{(i,j)\in\mathcal{E}} \|\dot{\mathbf{x}}_i - \dot{\mathbf{x}}_j\|_2^2 + \sum_{(i,j)\notin\mathcal{E}} \|\dot{\mathbf{x}}_i - \dot{\mathbf{x}}_j\|_2^2 = \sum_{(i,j)\in\mathcal{E}} \|\mathbf{x}_i - \mathbf{x}_j\|_2^2 + \sum_{(i,j)\notin\mathcal{E}} \|\mathbf{x}_i - \mathbf{x}_j\|_2^2 . \tag{3}$$

- In practice, instead of always tracking the original value $TPSD(\mathbf{X})$, we can maintain a constant TPSD value C across all layers, where C is a hyperparameter that could be tuned per dataset.

PairNorm

- The normalization can be done via a two-step approach where TPSD is rewritten as

$$\text{TPSD}(\tilde{\mathbf{X}}) = \sum_{i,j\in[n]} \|\tilde{\mathbf{x}}_i - \tilde{\mathbf{x}}_j\|_2^2 = 2n^2 \left(\frac{1}{n}\sum_{i=1}^{n} \|\tilde{\mathbf{x}}_i\|_2^2 - \left\| \frac{1}{n}\sum_{i=1}^{n} \tilde{\mathbf{x}}_i \right\|_2^2 \right). \tag{4}$$

- To simplify the computation, we subtract the row-wise mean from each $\tilde{\mathbf{x}}_i$, i.e., $\tilde{\mathbf{x}}_i^c = \tilde{\mathbf{x}}_i - \frac{1}{n}\sum_i^n \tilde{\mathbf{x}}_i$ where $\widetilde{\mathbf{x}}_i^C$ denotes the centered representation. Note that this shifting does not affect the TPSD, and furthermore drives the term $\left\|\frac{1}{n}\sum_{i=1}^{n} \tilde{\mathbf{x}}_i\right\|_2^2$ to zero, where computing $\text{TPSD}(\tilde{\mathbf{X}})$ boils down to calculating the squared Frobenius norm of $\tilde{\mathbf{X}}^c$ and overall takes $O(nd)$. That is,

$$\text{TPSD}(\tilde{\mathbf{X}}) = \text{TPSD}\left(\tilde{\mathbf{X}}^c\right) = 2n \left\|\tilde{\mathbf{X}}^c\right\|_F^2 \tag{5}$$

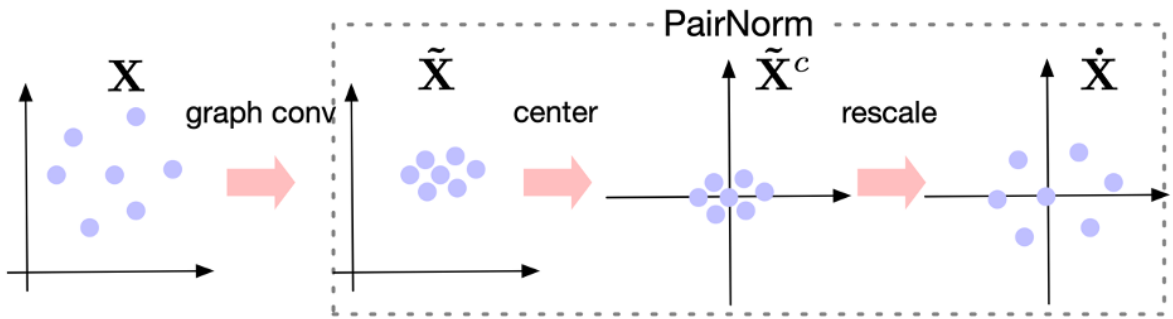
PairNorm

- In summary, PairNorm can be written as a two-step, center-and-scale, normalization procedure:

$$\begin{aligned}\tilde{\mathbf{x}}_i^c &= \tilde{\mathbf{x}}_i - \frac{1}{n}\sum_{i=1}^{n}\tilde{\mathbf{x}}_i \\ \dot{\mathbf{x}}_i &= s \cdot \frac{\tilde{\mathbf{x}}_i^c}{\sqrt{\frac{1}{n}\sum_{i=1}^{n}\left\|\tilde{\mathbf{x}}_i^c\right\|_2^2}} = s\sqrt{n} \cdot \frac{\tilde{\mathbf{x}}_i^c}{\sqrt{\left\|\tilde{\mathbf{X}}^c\right\|_F^2}}\end{aligned} \tag{6}$$

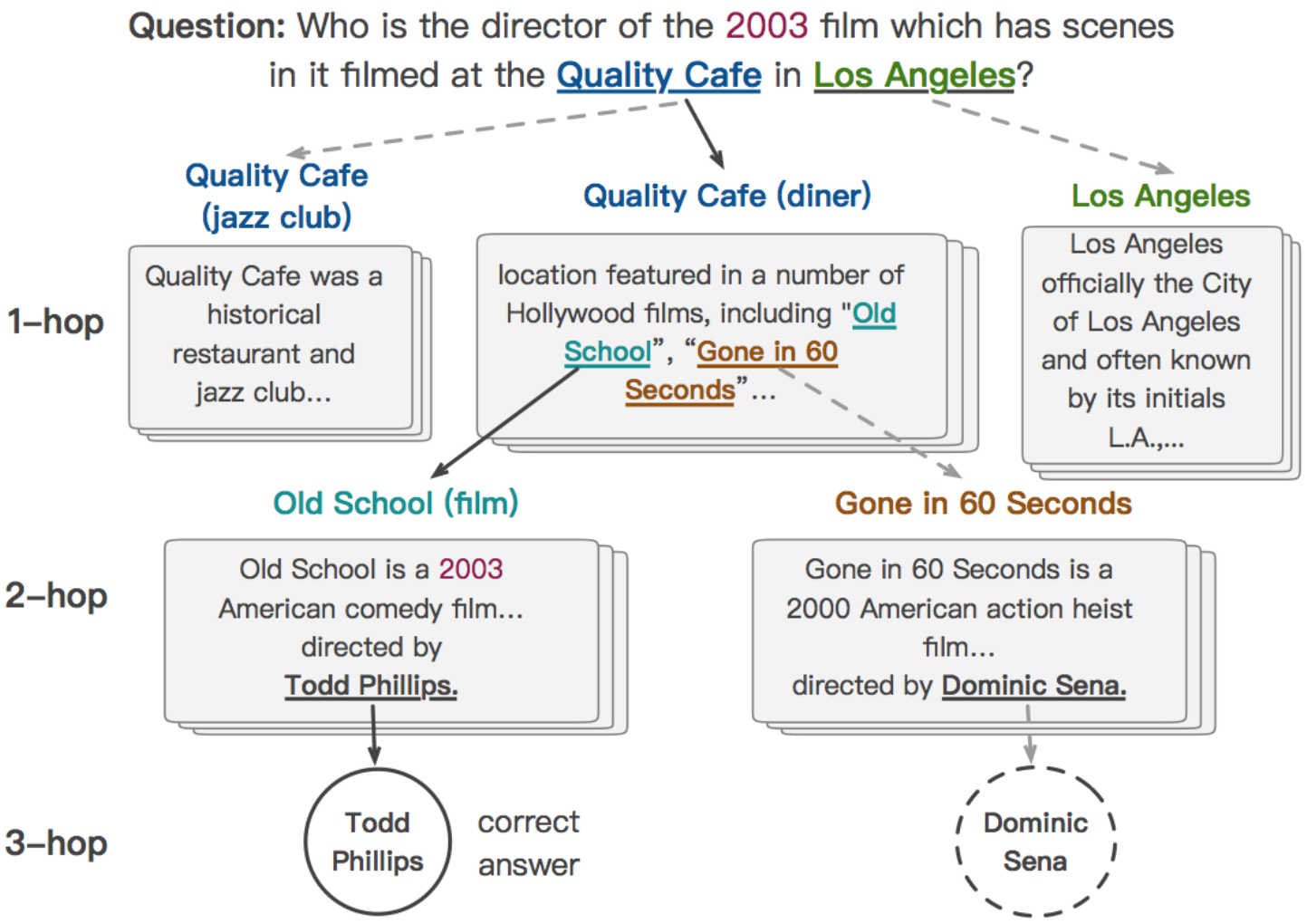
  where s is a hyperparameter that determines C.

- An illustration of PairNorm is given as follows.



GCN - One Application

Cognitive Graph for Multi-hop Reading Comprehension at Scale

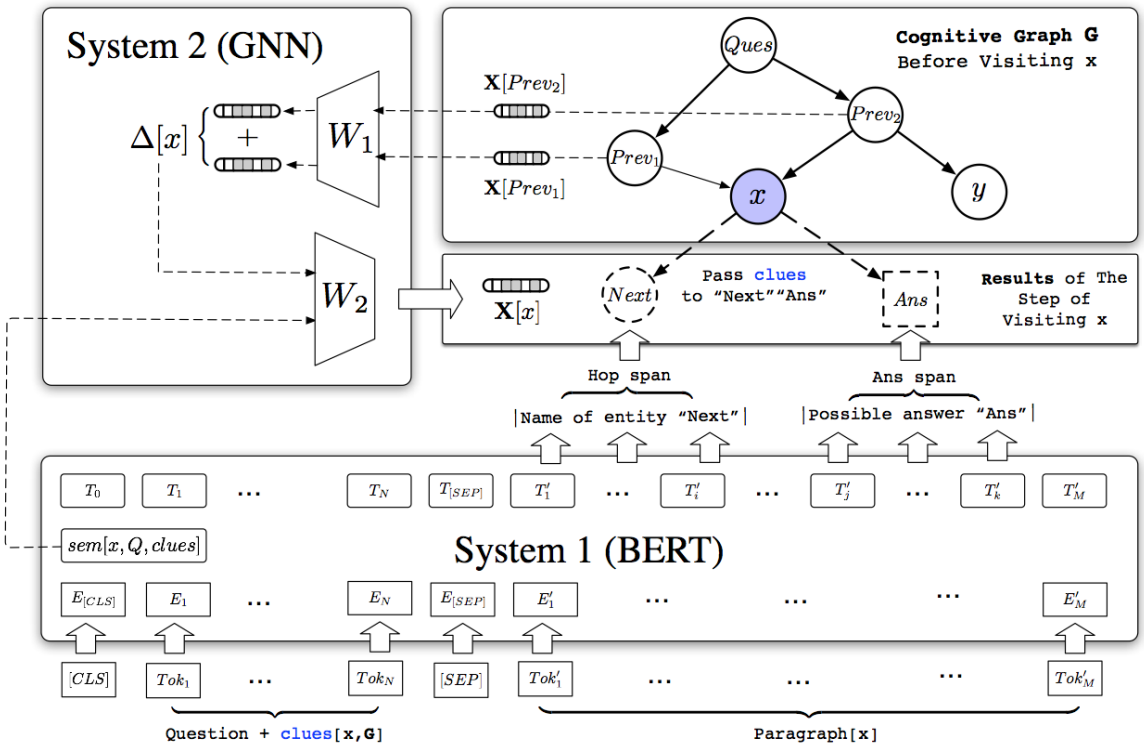


Cognitive Graph QA

Algorithm 1: Cognitive Graph QA

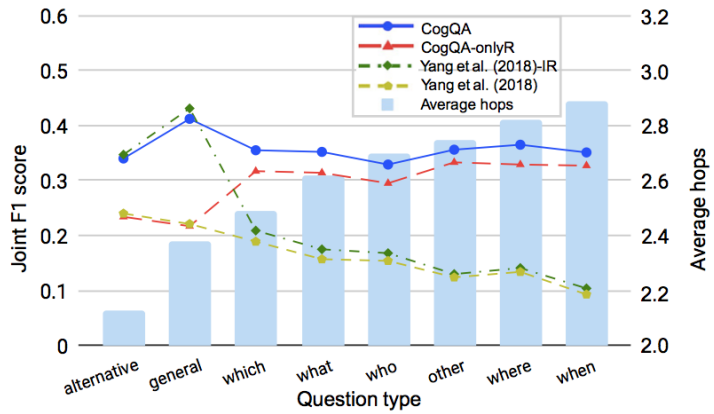
Input:
System 1 model $\mathcal{S}_1$, System 2 model $\mathcal{S}_2$,
Question Q, Predictor $\mathcal{F}$,Wiki Database $\mathcal{W}$

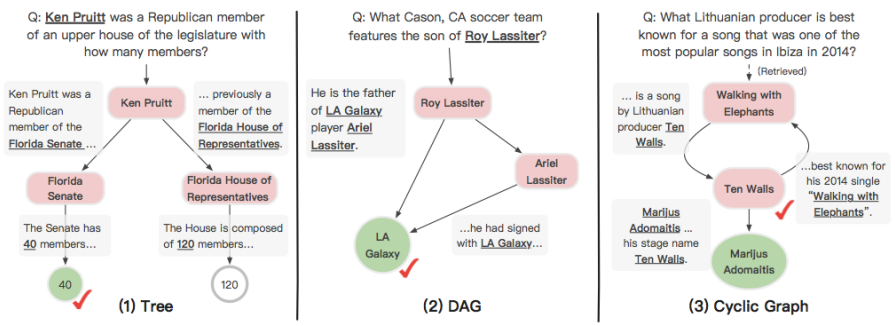
Initialize cognitive graph $\mathcal{G}$ with entities mentioned in Q
Initialize the frontier nodes as all nodes in $\mathcal{G}$
repeat
  pop a node x from frontier nodes
  locate $para[x]$ in $\mathcal{W}$
  collect $clues[x, \mathcal{G}]$ from predecessor nodes of x
  generate $sem[x, Q, clues]$ with $\mathcal{S}_1$
  if *x is a hop node* **then**
    find hop and answer spans in $para[x]$ with $\mathcal{S}_1$
    for *y in spans* **do**
      if $y \in \mathcal{G}$ *and* $edge(x, y) \notin \mathcal{G}$ **then**
        add y to frontier nodes
      else if $y \notin \mathcal{G}$ *and* $y \in \mathcal{W}$ **then**
        create a new frontier node for y
      end
      add edge (x, y) to $\mathcal{G}$
    end
  end
  update hidden representation $\mathbf{X}$ with $\mathcal{S}_2$
until *there is no frontier node in* $\mathcal{G}$;
Return $\arg\max_{\text{answer node } x} \mathcal{F}(\mathbf{X}[x])$



Results

	Model	Code	Ans		Sup		Joint	
			EM	F_1	EM	F_1	EM	F_1
1 Feb 21, 2019	Cognitive Graph (single model) *Anonymous*		**37.12**	**48.87**	**22.82**	**57.69**	**12.42**	**34.92**
2 Mar 5, 2019	MUPPET (single model) *Anonymous*		30.61	40.26	16.65	47.33	10.85	27.01
3 Mar 4, 2019	GRN (single model) *Anonymous*		27.34	36.48	12.23	48.75	7.40	23.55
4 Nov 25, 2018	QFE (single model) *NTT Media Intelligence Laboratories*		28.66	38.06	14.20	44.35	8.69	23.10
5 Oct 12, 2018	Baseline Model (single model) *Carnegie Mellon University, Stanford University, & Universite de Montreal* (Yang, Qi, Zhang, et al. 2018)		23.95	32.89	3.86	37.71	1.85	16.15
- Feb 28, 2019	DecompRC (single model) *Anonymous*		30.00	40.65	N/A	N/A	N/A	N/A
- Mar 3, 2019	MultiQA (single model) *Anonymous*		30.73	40.23	N/A	N/A	N/A	N/A





Content

1. Introduction
2. Node Embeddings
3. Graph Neural Networks

Thanks.

HP: http://keg.cs.tsinghua.edu.cn/jietang/
Email: jietang@tsinghua.edu.cn