Graph Embedding Learning

Special topics on Machine Learning

Fall 2018, NTU CSIE

Survey Papers:
1. Graph Embedding Techniques, Applications, and Performance: A Survey, IEEE PAMI, by Palash Goyal, Emilio Ferrara
2. A Comprehensive Survey of Graph Embedding: Problems, Techniques and Applications, by Hongyun Cai, Vincent W. Zheng, Kevin Chen-Chuan Chang

Outline

- Introduction: What is Graph Embedding Learning?
- Node Embedding Methods
  - Matrix Factorization approaches
  - Random Walk approaches
  - Deep neural networks
- State-of-the-art Models
  - PRUNE
  - Graph convolutional networks
- Future Research Topics

Introduction

- What is graph embedding learning?
- Given a graph, we want to describe the information as vector representations (also called embeddings).
- For a typical homogeneous graph $G = (V, E)$, containing the vertex (node) set V and the edge (link) set E, the graph embedding includes:
  - Node embeddings (e.g., user entity, publication)
  - Edge embeddings (e.g., friendship, co-authorship)
- For the remaining lecture, we will focus on the node embedding learning.

Different Titles, Same Topic

- Distributed Large-scale Natural Graph Factorization
- DeepWalk: Online Learning of Social Representations
- LINE: Large-scale Information Network Embedding
- GraRep: Learning Graph Representations with Global Structural Information
- node2vec: Scalable Feature Learning for Networks
- Asymmetric Transitivity Preserving Graph Embedding
- Try-Party Deep Network Representation

Node embedding

- **Goal**: to learn a vector representation for each node.

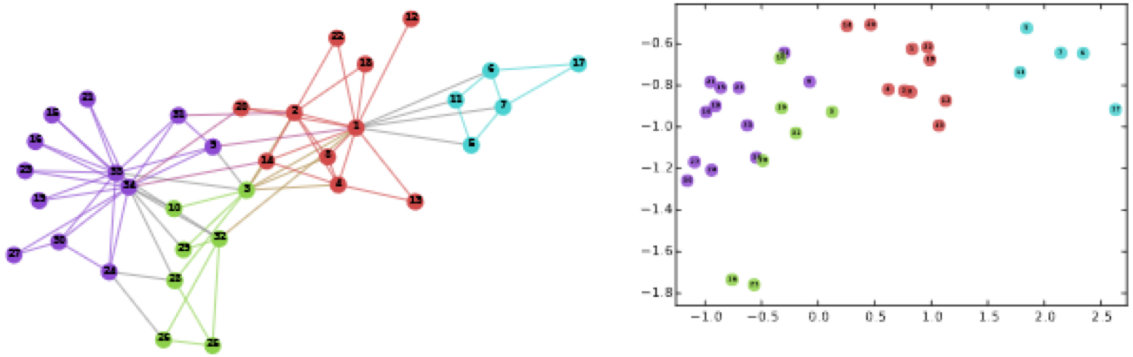
- What do vector representations actually mean?
- The graph structure itself contains certain properties and information (e.g., the characteristics of each nodes), and we want to leverage those information.
- However, it is relatively difficult to directly use the graph structure on further learning tasks.
- Therefore, node embedding becomes useful for further machine learning tasks.

Node embedding (cont.)

- **Another view**: graph structures are complicated, and nodes contain high-dimensional information (e.g., *k*-degree proximity, clustering properties), which is hard to describe.
- We can consider the node embedding learning process as a dimension reduction (or feature extraction).
  - Recall: PCA on high-dimensional raw features, but now we need corresponding methods on graphs.
  - A node in a specific graph => a *K*-dimensional vector.
- Also, it provides better flexibility for further usages.
  - Most machine learning and data mining models take vectors as inputs.

Key Questions to Answer

- For the node embedding learning, there are some main questions:
  - What information do we want to preserve through the output embeddings?
  - How to preserve it?
  - How to use embeddings (what kind of applications)?



(figure from DeepWalk)

Applicable tasks

- Node classification
- Regression on links
- Community detection
- Link prediction
- Graph visualization
- Ranking nodes
- …

What information to preserve?

- The key point to determine which kinds of node embeddings we want is based on the applications.
  - Different applications may require different information.
- Common goals of most works:
  - **Proximity**: Two nodes that are closer in the graph shall share similar embeddings.
  - **Community** (meta-proximity): Two nodes in the same community (cluster) shall share similar embeddings
  - **Structure characteristics**: Two nodes act as similar characters in similar subgraph structures shall have similar embeddings. (e.g., struc2vec)

Naïve strategies

- Intuitively, we can consider explicit features (which are obvious or easy to be computed) of nodes as representations.
  - For example, node degrees, clustering coefficients.
- These explicit node features were useful for conventional approaches tackling tasks on graphs.
- However, it is hard to cover all neede information based on hand-crafted features.
  - If the handcrafted features cannot carry all the information of nodes => information loss.

In this course

- To obtain better vector representations, modern node embedding approaches consider to learn proper vectors via designated learning objectives.
  - Usually as unsupervised learning settings.
  - i.e., graph => embeddings => tasks using embeddings
- In this course, we will first introduce common learning models for effective node embeddings.
- Then, the state-of-the-art methods will be discussed.

Outline

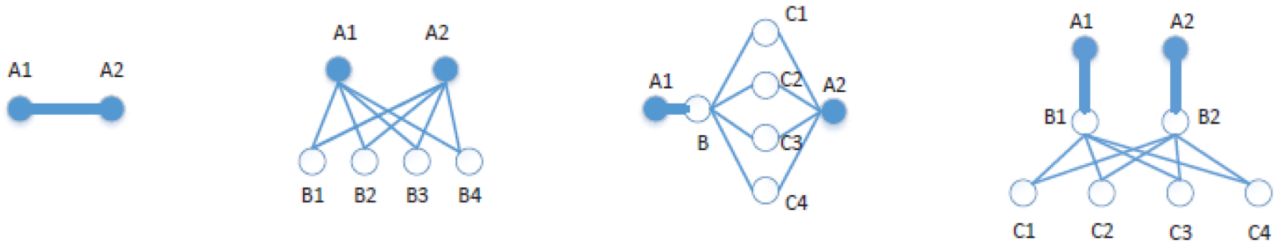
- Introduction: What is Graph Embedding Learning?
- Node Embedding Methods
  - Matrix Factorization approaches
  - Random Walk approaches
  - Deep neural networks
- State-of-the-art Models
  - PRUNE
  - Graph convolutional networks
- Future Research Topics

How to Preserve: A general Framework

- Model it as an optimization task
  - Goal: Adjust Embeddings (U_i) to optimize certain objective
- Training instances are obtained directly from the graph, e.g.
  - H: set of connected node pairs (i, j) where $\text{similarity}(U_i, U_j)$ should be high
  - L: set of disconnected node pairs (i, j) where $\text{similarity}(U_i, U_j)$ should be low
- Different models might consider different $measures\ of\ proximity$
  - e.g, $\text{similarity}(U_i, U_j) = U_i^\top U_j$: inner product

A general Framework: Training

- Positive instance set H
  - which $(i, j) \in H$ varies in various of works
- Negative instance set $L \subseteq (\mathrm{V} \times V) \setminus H$
  - Quadratic!! Normally require to sample negative instances to reduce time complexity
- Commonly use similarity matrix $S \in \mathbb{R}^{N \times N}$ of node pairs
  - e.g. number of common neighbors
  - Models differe from how S is concstructed
- p-th order proximity
  - Topology and similarity between node pair (i, j) in distance p
  - Example: node pair (A_1, A_2) in 1,2,3,4th-order proximity



Cao, Shaosheng, Wei Lu, and Qiongkai Xu. "Grarep: Learning graph representations with global structural information." *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*. ACM, 2015.

Examples

- Task formulation
  - Classification
    - $U = \arg\max_{U} \prod_{(i,j)\in H} f\left(U_i^\top U_j\right) \prod_{(i,j)\in L} \left(1 - f\left(U_i^\top U_j\right)\right)$
    - Probability function $f: \mathbb{R} \to [0,1]$
      - $f(x) \in \{\frac{1}{1+e^{-x}}, \frac{e^x}{\sum_y e^y}\}$: sigmoid or softmax
  - Regression
    - $U = \arg\min_{U} \sum_{(i,j)} \left(U_i^\top U_j - S_{ij}\right)^2$ or $\mathrm{KL}(S || f(U^\top U))$
    - Matrix factorization (MF) or SVD

Node embedding learning

- The modern models learning node embeddings can basically be summarized into three categories:
  - (1) Matrix factorization methods
  - (2) Random walk methods
  - (3) Deep neural networks

- Different types of methods solve the problem (preserving the information of nodes) in different aspects and styles.

Matrix Factorization Methods

- Matrix factorization (MF) is a commonly used strategy to learn low-dimensional representations and has achieved numerous successes among applications (e.g., recommender systems).
- The main idea of MF is to learn a corresponding low-rank approximation with respect to the given input matrix.
- Variants of MF:
  - Non-negative MF
  - Symmetric MF

Symmetric MF Method

- One of the most straightforward methods to conduct MF for node embedding learning is to leverage the adjacency matrix (to preserve 1st order proximity).
- Let z_a and z_b be the node embedding vectors of two vertices a and b.
- Suppose that
  - $z_a^{\mathrm{T}} z_b = 1$, if a and b are directly connected.
  - $z_a^{\mathrm{T}} z_b = 0$, otherwise.
- Then we have $z_a^{\mathrm{T}} z_b = \mathbf{S}_{ab}$, where $\mathbf{S}$ is the adjacency matrix.

Symmetric MF Method (cont.)

- Since $z_a^{\mathrm{T}} z_b = \mathbf{S}_{ab}$, now we can model the node embedding problem as a symmetric MF problem:

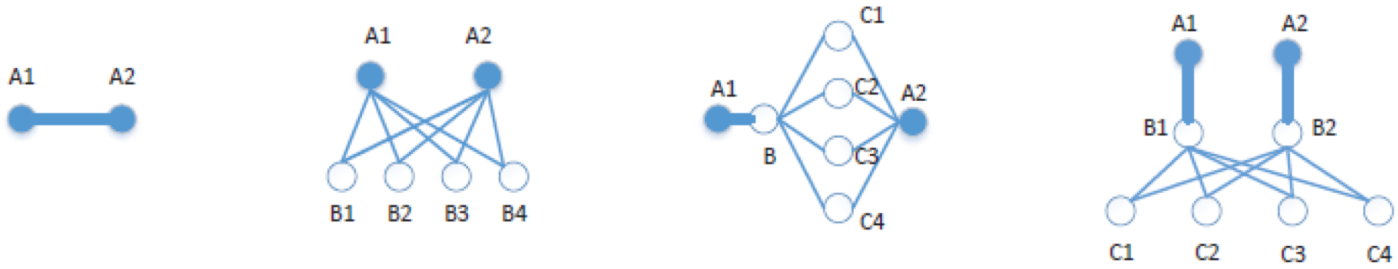
$$\mathrm{argmin}_{\mathbf{Z}} \quad \| \mathbf{Z}^{\mathrm{T}}\mathbf{Z} - \mathbf{S} \|^2_F + \lambda \| \mathbf{Z} \|^2$$

(an exmaple using L2 Frobenius loss and L2 regularization)

- Note that, matrix **S** can be directly derived from the given graph.
  - It is unsupervised. (No matter what successive learning tasks we want to use the embedding **Z**, we can always perform the same MF learning framework to obtain **Z)**.

GraRep

- Recall the main question of node embedding learning: “What to preserve?”
- For the basic MF method (previous pages), we simply aim at preserving 1st order proximity.
- **Idea**: representing higher-order proximities is important.



Cao, Lu, and Xu. "GraRep: Learning graph representations with global structural information." *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*. ACM, 2015.

GraRep (cont.)

- How to capture high-order proximities?
- GraRep introduces the transition matrix $A = D^{-1}S$.
- S is the adjacency matrix.
- D is the diagonal (degree) matrix

$$D_{ij} = \begin{cases} \sum_p S_{ip}, & \text{if } i = j \\ 0, & \text{if } i \neq j \end{cases}$$

- Based on this definition, we have the interpretations:
  - A_{ij}: Probability from node i to node j via one step.
  - $(A^k)_{ij}$: Probability from node i to node j via k steps.

GraRep: Algorithm

GraRep Algorithm

Input
Adjacency matrix S on graph
Maximum transition step K
Log shifted factor β
Dimension of representation vector d

1. Get k-step transition probability matrix A^k
Compute $A = D^{-1}S$
Calculate $A^1, A^2, \ldots, A^K$, respectively
2. Get each k-step representations
For $k = 1$ to K
2.1 Get positive log probability matrix
calculate $\Gamma_1^k, \Gamma_2^k, \ldots, \Gamma_N^k$ ($\Gamma_j^k = \sum_p A_{p,j}^k$) respectively
calculate $\{X_{i,j}^k\}$

$$X_{i,j}^k = \log\left(\frac{A_{i,j}^k}{\Gamma_j^k}\right) - \log(\beta)$$

assign negative entries of X^k to 0
2.2 Construct the representation vector W^k
$[U^k, \Sigma^k, (V^k)^T] = SVD(X^k)$
$W^k = U_d^k (\Sigma_d^k)^{\frac{1}{2}}$
End for
3. Concatenate all the k-step representations
$W = [W^1, W^2, \ldots, W^K]$

Output
Matrix of the graph representation W

- First, we calculate the transition matrices A, A^2, …, A^K.
- Then based on A's, we can derive the "adjacency-like" matrix X^k for k-step extension.
  - (Derivation details are omitted here.)
- Similar to the 1-step version, we perform MF (SVD) on each X^k to get embedding W^k.
- After computing all the k-step node embeddings W^k ($k = 1$ to K), the final representation is the concatenation of them.

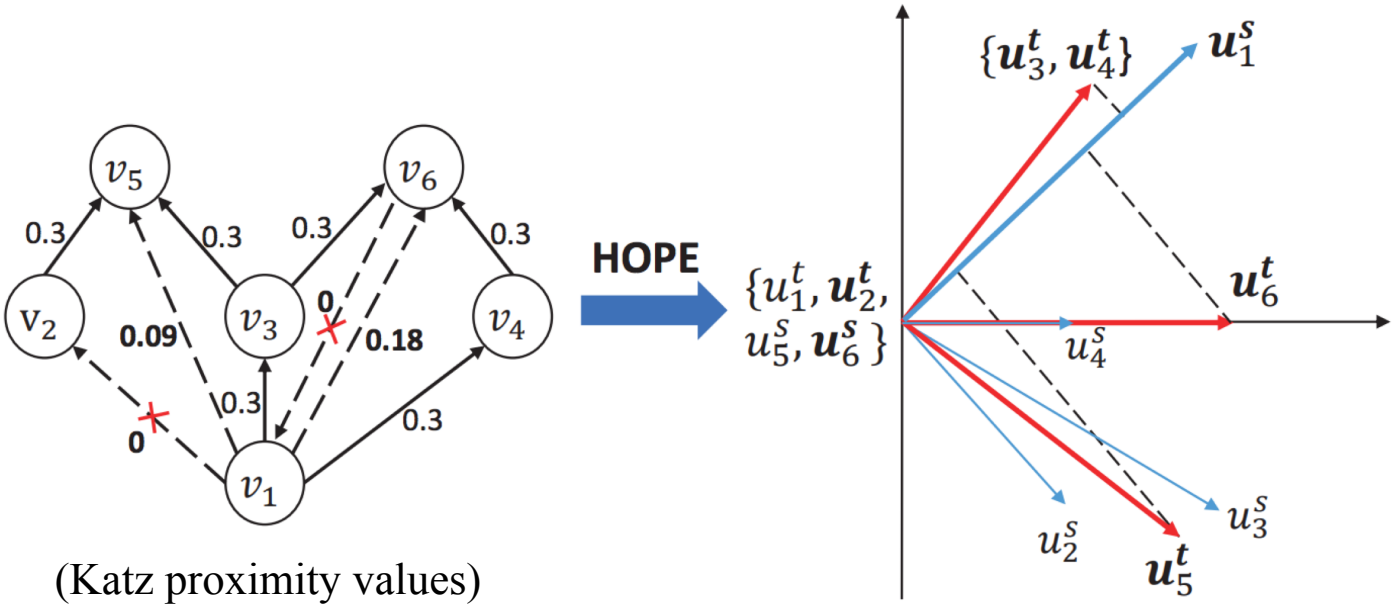
HOPE

- High-Order Proximity Preserved Embedding
- **Idea**: source node and destination node serve different roles in directed graphs.
  - z_a, z_b for <a, b> doesn't reveal whether a or b is a source
  - Solution: use two distinct embedding vectors to describe a single node: one for acting as source; the other one for acting as destination of a link.
- A framework of asymmetric transitivity preserving graph embedding.

Ou, Mingdong, et al. "Asymmetric transitivity preserving graph embedding." *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2016.

HOPE (cont.)

- An example of HOPE:
  - Each node i has two vectors ($\boldsymbol{u}_i^s$ and $\boldsymbol{u}_i^t$).
  - High-order proximity: v_1->v_3->v_6 & v_1->v_4->v_6 => v_1->v_6 .



(Katz proximity values)

HOPE (cont.')

- Problem formulation: Asymmetric MF optimization

$$\min \|\mathbf{S} - \mathbf{U}^s \cdot \mathbf{U}^{t\top}\|_F^2$$

  - $\mathbf{U}^s$: node embedding vectors when acting as sources.
  - $\mathbf{U}^t$: node embedding vectors when acting as dests.
  - The proximity matrix $\mathbf{S}$ is approximated by $S_{ij} = U^s_i{}^{\mathrm{T}}\, U^t_j$.

- HOPE also use approximations on high-order proximity measurements to handle large-scale networks.
  - Refer to the paper for details.

Outline

- Introduction: What is Graph Embedding Learning?
- Node Embedding Methods
  - Matrix Factorization approaches
  - Random Walk approaches
  - Deep neural networks
- State-of-the-art Models
  - PRUNE
  - Graph convolutional networks
- Future Research Topics

Random Walk Methods

- **Idea**: Reduction to word embedding.
  - We know how to generate effective word embedding of sentences nowadays (details next week).
  - Node sequences from random walks (RWs) have similar node-frequency distributions as sentences of a document.
  - Feed node sequences to word embedding models (e.g., word2vec) to generate network embeddings.
- Different from MF-based methods, RW-based approaches consider sampled paths of graph.

Random Walk Methods (cont.)

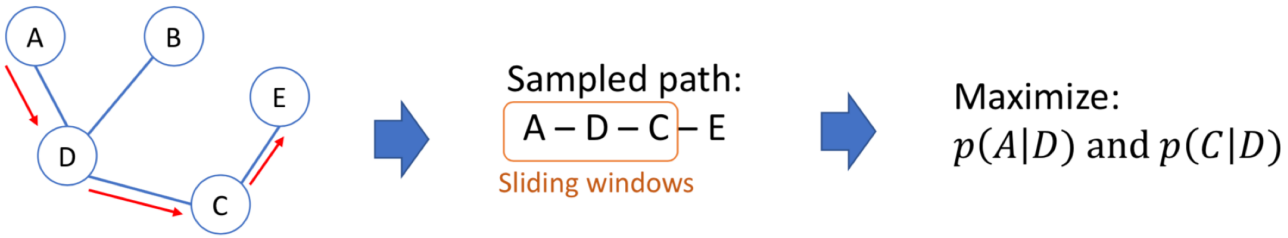
- **Goal**: Estimate P(node i | node j).
  - How likely can a walker reach node i from node j.
- General framework:
  - Let several random-walkers travel across the graph and collect paths.
  - Assumption: nodes co-occur in the path should have relatively high P(node i | node j).
  - Optimization model:

$$\mathrm{argmax}_\theta \sum_{(i,j)\in \mathrm{paths}} \left(\log \sigma(z_i^\top z_j) - \log \sum_x \sigma(z_i^\top z_x) \right)$$

  - Key question: "What kinds of paths should be sampled?"

DeepWalk

- Similar to applying word2vec on graphs.
- Working flow:
  - Decide path length k, usually no more than 10.
  - Uniformly sample a starting node (and then uniformly sample its neighbors) to generate paths.
    - The obtained paths act similar roles as sentences in text data.
  - Estimate $P(\text{node } i \mid \text{node } j)$ by skip-gram models.



Perozzi, Bryan, Rami Al-Rfou, and Steven Skiena. "Deepwalk: Online learning of social representations." *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2014.

node2vec

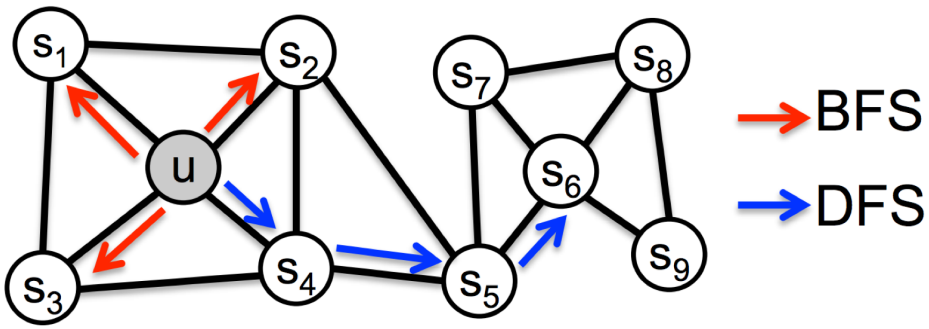
- node2vec provides a more flexible (compare to DeepWalk) walking mechanism to capture different types of information.
  - Perform both BFS and DFS strategies on random walks.
  - Second-order random walk with search bias α_{pq}.
    - p and q are two parameters guiding the walk (explained later).
  - Can enclose additional edge features if available.

- http://snap.stanford.edu/node2vec

Grover, Aditya, and Jure Leskovec. "node2vec: Scalable feature learning for networks."
Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2016.

node2vec (cont.)

- Two different strategies of random walk:
  - Breadth-first search (BFS)
    - Micro-view of the neighborhood
    - More effective for capturing local structure
  - Depth-first search (DFS)
    - Macro-view of the neighborhood
    - More effective for capturing global structure

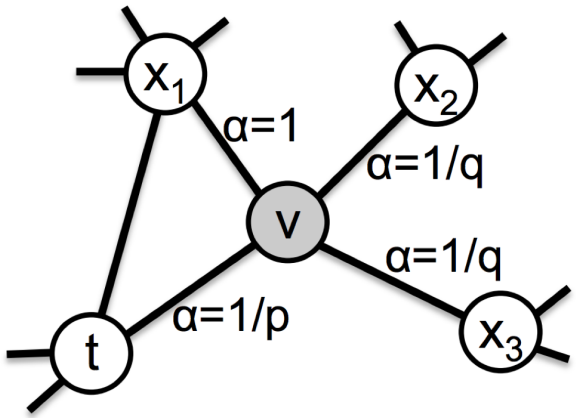


We want to mix both strategies!

node2vec (cont.')

- Second-order random walk with search bias α_{pq}:
  - RW with weighted probabilities based on the structure.
  - t : the previous node just visited
  - v : the current node
  - x_i : the remaining neighbor nodes
  - $d_{tx} = \{0, 1, 2\}$ denotes the shortest path distance between nodes t and x.

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



Relation between MF-based and RW-based approaches

- Although RW-based methods look very different to MF models, most of them can actually be described as solving MF with specific matrices.

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

Qiu, Jiezhong, et al. "Network Embedding as Matrix Factorization: Unifying DeepWalk, LINE, PTE, and node2vec." *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*. ACM, 2018.

Outline

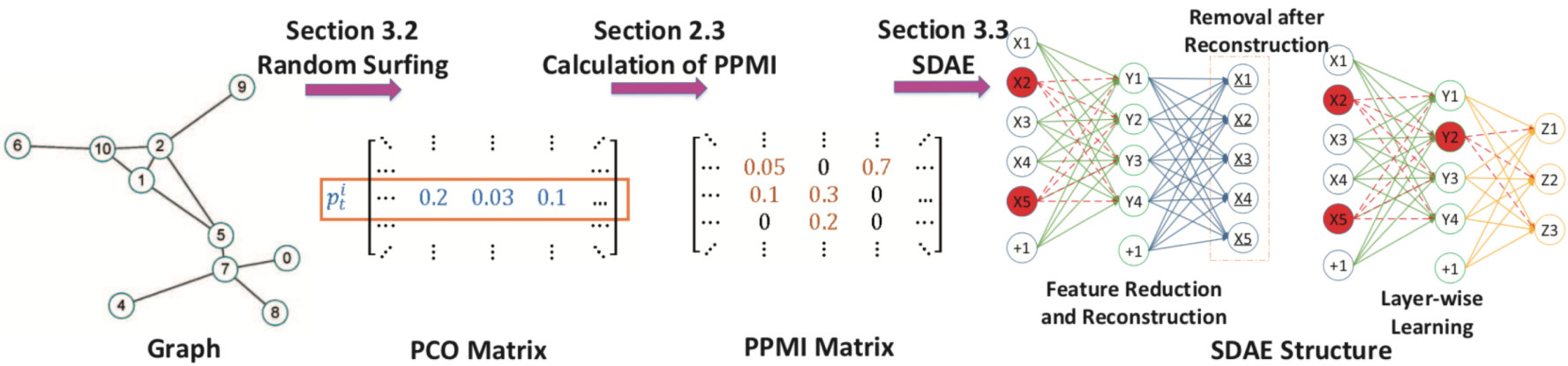
- Introduction: What is Graph Embedding Learning?
- Node Embedding Methods
  - Matrix Factorization approaches
  - Random Walk approaches
  - Deep neural networks
- State-of-the-art Models
  - PRUNE
  - Graph convolutional networks
- Future Research Topics

DNN-based Node Embedding

- Recently, models based on deep neural networks have shown superior ability on node embedding learning.
  - Tradeoff: heavier computation loads.
- DNN hidden layers as embeddings
  - Auto-encoders
  - Specified deep networks
  - Graph convolutional networks
    - Applying CNNs on Graph Laplacian for *supervised* tasks
    - will be separately discussed later

DNGR

- Deep Neural Networks for Learning Graph Representations:
  - Step 1: Random surfing gets a co-occurrence matrix.
  - Step 2: Calculate the PPMI matrix.
  - Step 3: Extract embedding with the *stacked denoised autoencoder* (SDAE) framework.



Cao, Shaosheng, Wei Lu, and Qiongkai Xu. "Deep Neural Networks for Learning Graph Representations." *AAAI*, 2016.

DNGR (cont.)

- For Step 1, random walk has several weaknesses:
  - Boundary nodes in a finite (e.g., *k*-length) sequence is hard to capture
  - Difficult to choose hyper-parameters.
- Therefore, personalized PageRank with restart (p_0) is adopted.
  - Starts from a specific node *v* ('personalized'), estimate the probability to stop at each node *i* in the end:
  - p_k is an n-dimensional vector representing the probability of reaching node *i* from *v* in k-step, A is the transitional matrix

$$p_k = \alpha \cdot p_{k-1} A + (1 - \alpha) p_0$$

- Sum the score of first *K* steps as the 'co-occurance' of *v* and each node *i*.

DNGR (cont.')

- Instead of directly use the co-occurrence matrix we just obtained, DNGR considers the corresponding PPMI matrix for the later operations (Step 2).
- PMI: Pointwise mutual information

$$PMI_{v,i} = \log\left(\frac{p_{v,i}}{p_v p_i}\right)$$

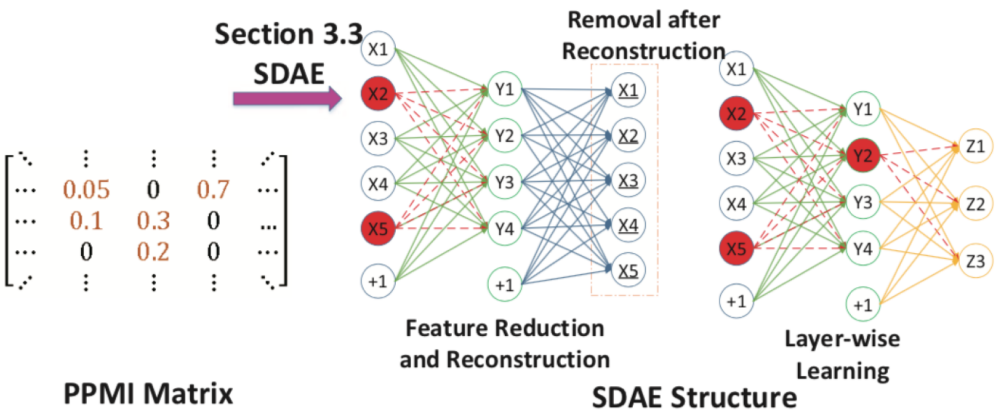
  - PMI matrix has been proven to provide better word representations (Bullinaria and Levy, 2007).
  - PPMI: positive PMI value

$$X_{w,c} = max(PMI_{w,c}, 0)$$

Bullinaria, John A., and Joseph P. Levy. "Extracting semantic representations from word co-occurrence statistics: A computational study." *Behavior research methods* 39.3 (2007): 510-526.

DNGR (cont.”)

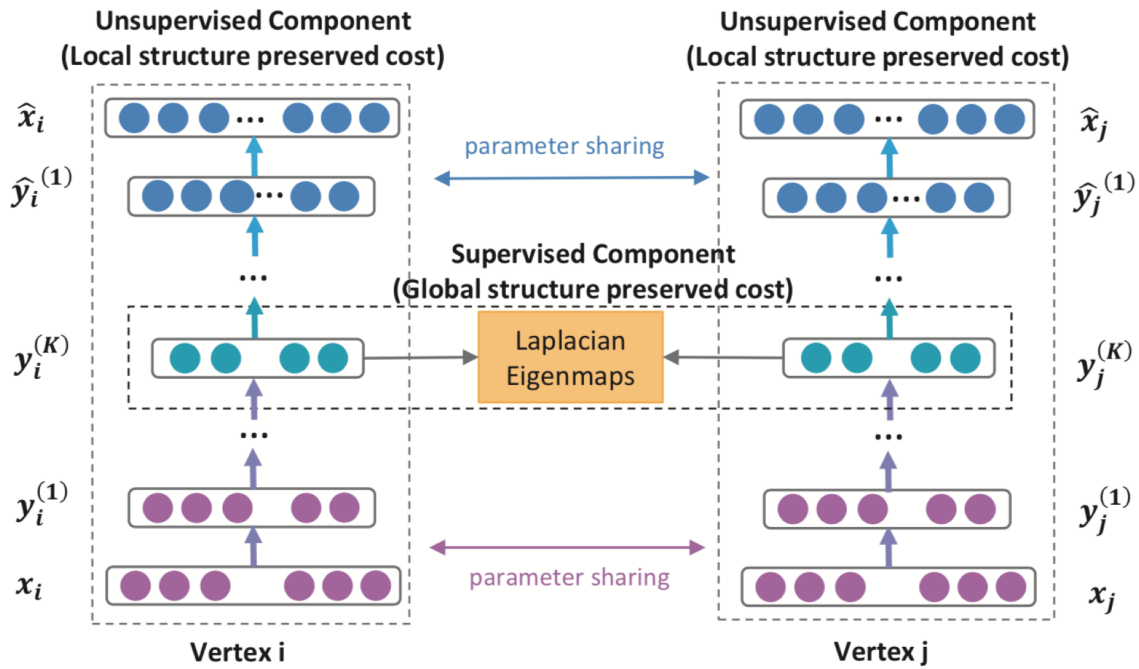
- Step 3: with the PPMI matrix, the stacked denoised autoencoder (SDAE) framework is conducted.
  - SDAE can be considered as a normal AE with dropout in different layers
- Similar to standard autoencoders, data (PPMI values) are reconstructed from the latent representations.



$$\min_{\theta_1,\theta_2} \sum_{i=1}^{n} L\left(x^{(i)}, g_{\theta_2}(f_{\theta_1}(\tilde{x}^{(i)}))\right)$$

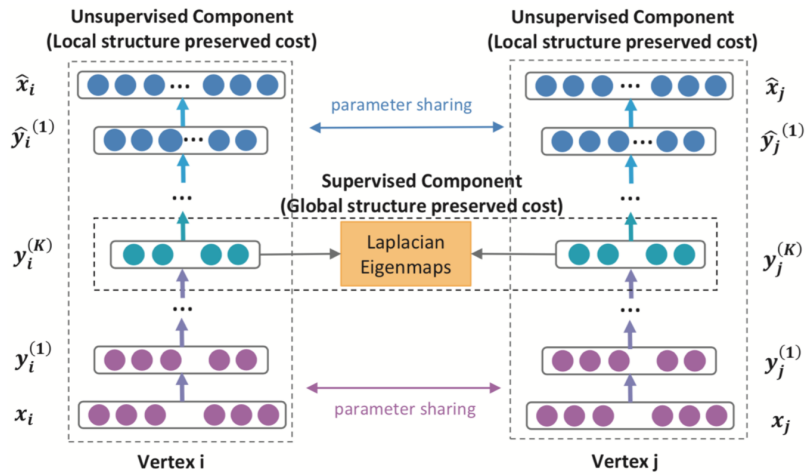
SDNE

- Structural Deep Network Embedding
- Preserving both 1st order and 2nd order proximity



Wang, Daixin, Peng Cui, and Wenwu Zhu. "Structural deep network embedding.“, *ACM SIGKDD*, 2016.

SDNE (cont.)



- The loss function combine the loss from 1st order (L_{1st}) and the loss from 2nd order proximity (L_{2nd}):

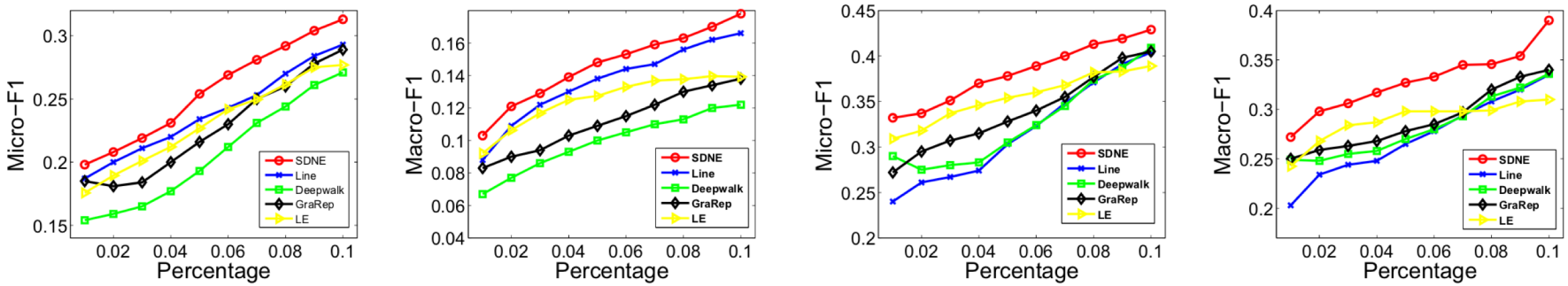
$$\mathcal{L}_{mix} = \mathcal{L}_{2nd} + \alpha\mathcal{L}_{1st} + \nu\mathcal{L}_{reg}$$

$$= \|(\hat{X} - X) \odot B\|_F^2 + \alpha \sum_{i,j=1}^{n} s_{i,j} \|\mathbf{y}_i - \mathbf{y}_j\|_2^2 + \nu\mathcal{L}_{reg}$$

  - L_{reg}: the regularization term preventing overfitting.
  - S: the adjacency matrix ($s_{i,j} = \{0, 1\}$)
  - $\boldsymbol{y}_i$: the node embedding of node i.
  - B: if $s_{i,j} = 0$, $b_{i,j} = 1$; otherwise, b = $\beta > 1$.

SDNE: Performances

- As the experiment results shown, SDNE, which is a *DNN-based* model, can outperform the classic MF or RW-based methods.



(a) Micro-F1 and Macro-F1 on FLICKR (b) Micro-F1 and Macro-F1 on YOUTUBE

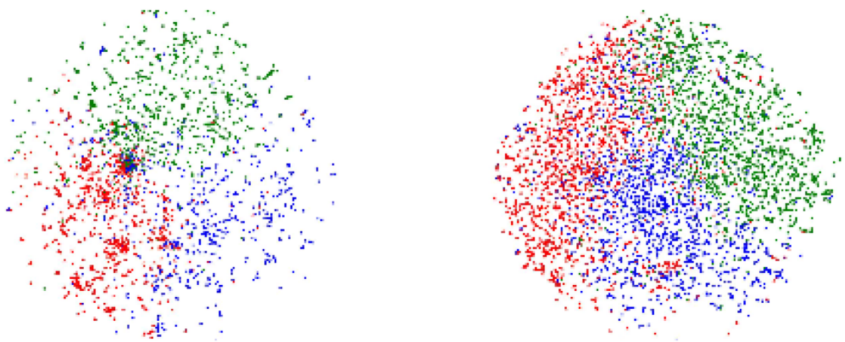
Table 4: MAP on ARXIV-GRQC and BLOGCATALOG on reconstruction task

Method	ARXIV-GRQC					BLOGCATALOG				
	SDNE	*GraRep*	*LINE*	*DeepWalk*	*LE*	*SDNE*	*GraRep*	*LINE*	*DeepWalk*	*LE*
MAP	**0.836****	0.05	0.69	0.58	0.23	**0.63****	0.42	0.58	0.28	0.12

Significantly outperforms GraRep at the: ** 0.01 level.

Other Models

- Other optimization goals are existed to preserve network structures.
- Examples
  - LINE [1]: minimization of KL divergence
  - NRCL [2]: minimization of Huber loss



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

(figure from LINE paper)

[1] Tang, Jian, et al. "Line: Large-scale information network embedding." *Proceedings of the 24th International Conference on WWW*, 2015.
[2] Wei, Xiaokai, et al. "Cross View Link Prediction by Learning Noise-resilient Representation Consensus." *Proceedings of the 26th International Conference on World Wide Web*, 2017.

Outline

- Introduction: What is Graph Embedding Learning?
- Node Embedding Methods
  - Matrix Factorization approaches
  - Random Walk approaches
  - Deep neural networks
- State-of-the-art Models
  - PRUNE
  - Graph convolutional networks
- Future Research Topics

PRUNE: Preserving Proximity and Global Ranking for Network Embedding

Yi-An Lai, Chin-Chi Hsu, Wen Hao Chen,
Mi-Yen Yeh and Shou-De Lin

NIPS 2017

Lai, Yi-An, et al. "PRUNE: Preserving Proximity and Global Ranking for Network Embedding."
Advances in Neural Information Processing Systems. 2017.

What to Preserve?

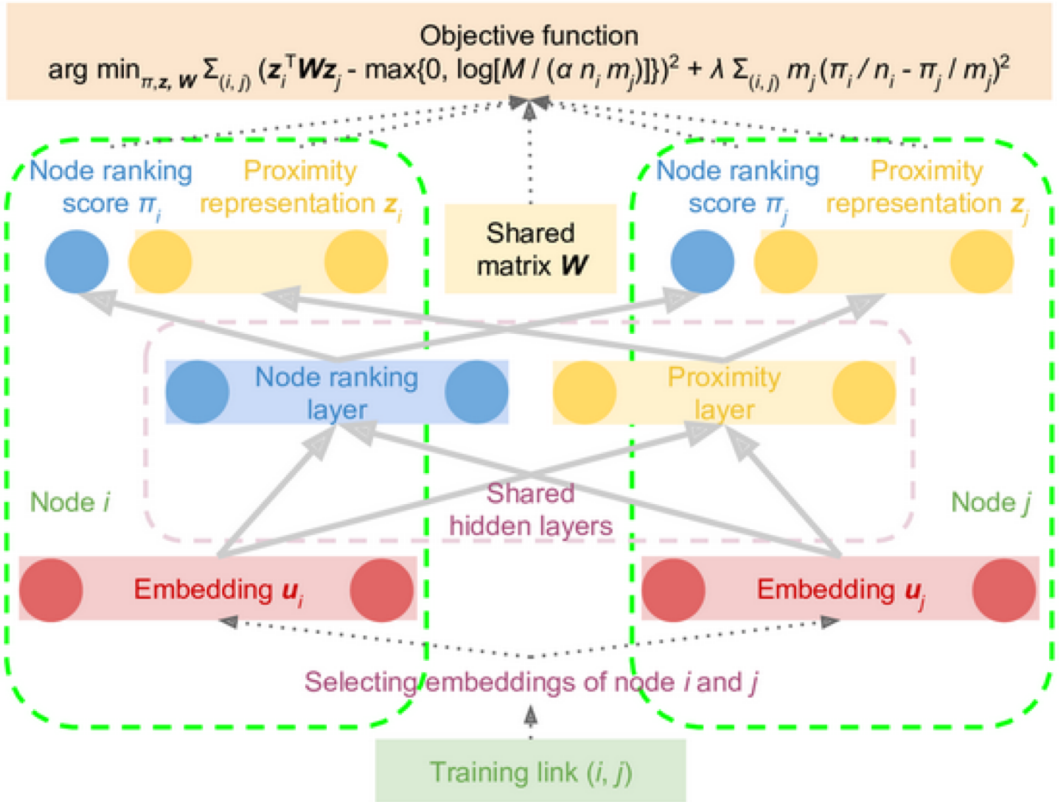
- Proximity (same as previous works): nodes with smaller shortest path distance should possess similar embeddings than those with longer shortest path.
- An additional goal: to preserve node ranking of importance in a graph.
  - The information of the importance of nodes is embedded in the network (i.e., PageRank)

Model Design

- Recall: common model designs:
  - Singular Value Decomposition or other Matrix Factorization formulations (e.g., GraRep, HOPE)
  - Random Walks (e.g., DeepWalk, node2vec)
  - Deep Neural Networks (e.g., DNGR, SDNE)

- Different from the previous works, PRUNE consider multi-task Siamese neural networks.

A generative model for node embedding learning



- Objective function:

$$\arg\min_{\pi\geq 0, z\geq 0, W\geq 0} \boxed{\overbrace{\underbrace{\sum_{(i,j)\in E}\left(\boldsymbol{z}_i^\top \boldsymbol{W}\boldsymbol{z}_j - \max\left\{0, \log\frac{M}{\alpha m_j n_i}\right\}\right)^2}_{\text{Matrix tri-factorization}}}^{\text{Proximity}}} + \lambda \overbrace{\underbrace{\sum_{(i,j)\in E} m_j\left(\frac{\pi_j}{m_j} - \frac{\pi_i}{n_i}\right)^2}_{\text{PageRank upper bound}}}^{\text{Node ranking}}$$

Proximity Preservation

- Simple idea: linked entities ⇔ similar embeddings
  - Original Goal: Sample α negative samples, apply binary logistic regression model:

$$\arg\max_{\boldsymbol{z},\boldsymbol{W}} \mathbb{E}_{(i,j)\in E}\left[\log \sigma(\boldsymbol{z}_i^\top \boldsymbol{W} \boldsymbol{z}_j)\right] + \alpha \mathbb{E}_{(i,k)\in F}\left[\log\left(1-\sigma(\boldsymbol{z}_i^\top \boldsymbol{W} \boldsymbol{z}_k)\right)\right]$$

  - Closed-form solution by derivative: ($y_{ij} = z_i^T W z_j$)

$$y_{ij} = \log \frac{M}{\alpha n_i m_j} = \log \frac{p_{s,t}(i,j)}{p_s(i)p_t(j)} - \log \alpha$$

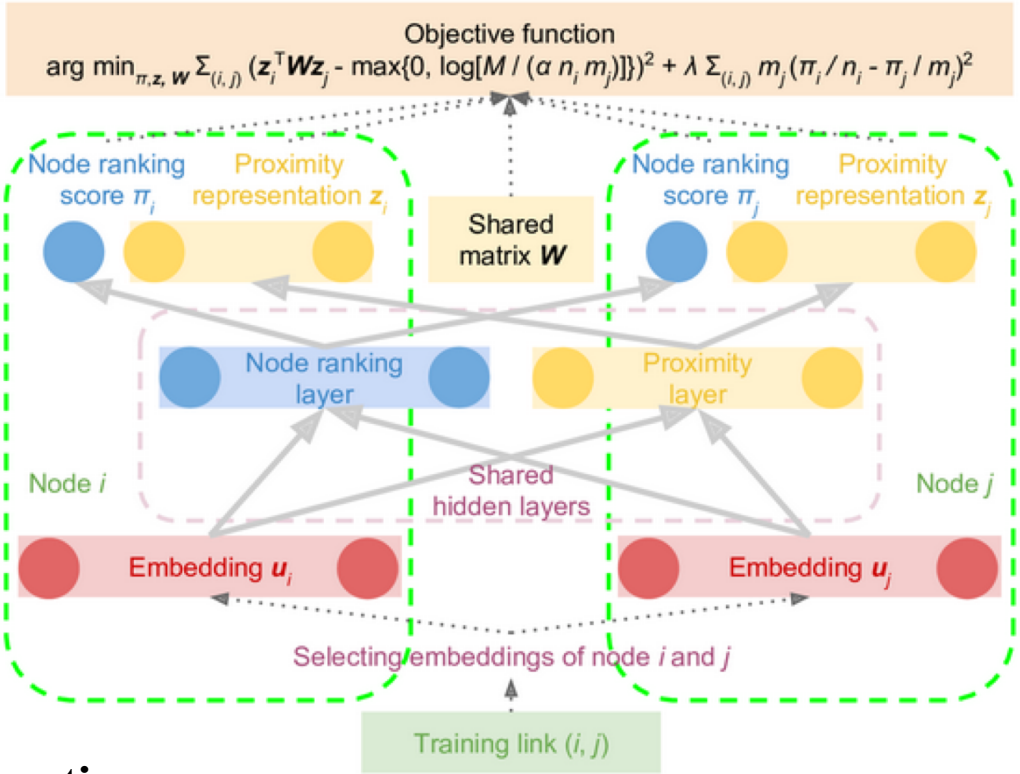
  - Pointwise mutual information (PMI) shifted by $\log \alpha$.
  - Goal: **minimize the gap between two sides rather than maximize the original objective** (No need for negative sampling!)

Tri-factorization: Why $z_i^{\mathrm{T}} W z_j$?

- Asymmetric transitivity:
  - $z_i^T z_j$ doesn't handle directed link but $z_i^T W z_j$ does
- Preserving both 1st and 2nd-order proximity
  - 2nd-order proximity: similar sets of direct predecessors and successors implies similar embeddings
  - existence of two links $(i,j)(z_i^T W z_j)$, $(j,k)(z_j^T W z_k)$ increase the likelihood of $(i,k)(z_i^T W z_k)$
- PMI matrix tri-factorization → Community interaction:
  - Can be regard as the expectation of community interactions

$$\boldsymbol{z}_i^\top \boldsymbol{W} \boldsymbol{z}_j \propto \mathbb{E}_{(i,j)}[\boldsymbol{W}] = \sum_{c=1}^{D} \sum_{d=1}^{D} \Pr(i \in C_c) \Pr(j \in C_d) w_{cd}$$

PRUNE: Framework



- Objective function:

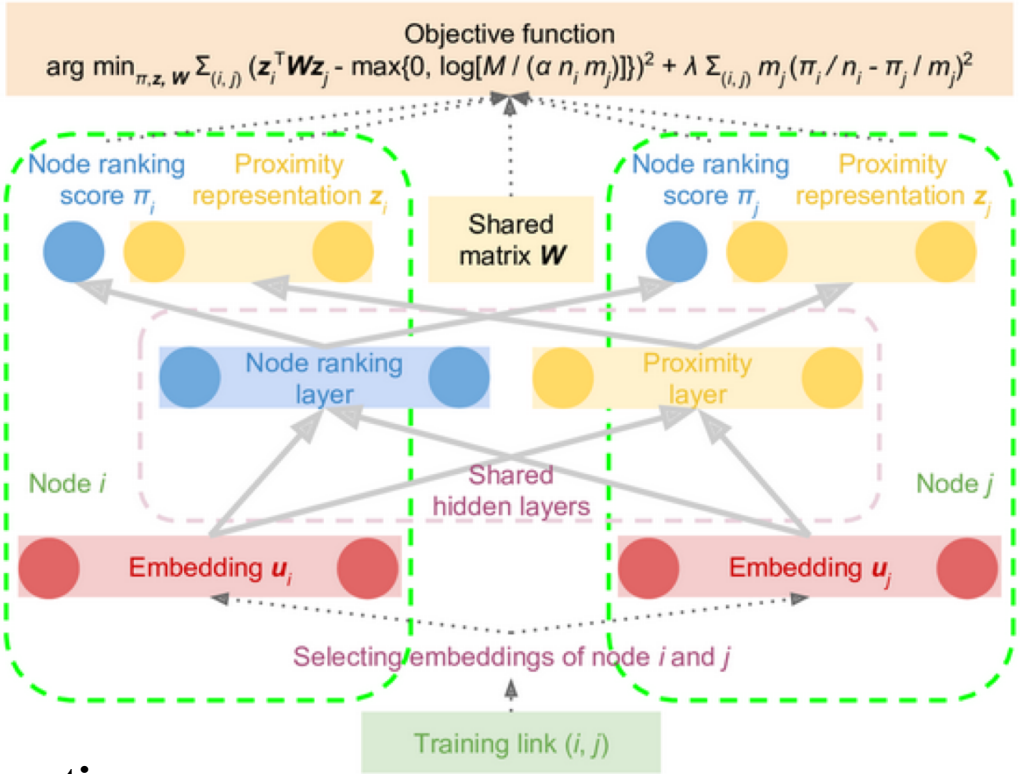
$$\arg\min_{\pi\geq 0, z\geq 0, W\geq 0} \overbrace{\underbrace{\sum_{(i,j)\in E}\left(\boldsymbol{z}_i^\top \boldsymbol{W}\boldsymbol{z}_j - \max\left\{0, \log\frac{M}{\alpha m_j n_i}\right\}\right)^2}_{\text{Matrix tri-factorization}}}^{\text{Proximity}} + \boxed{\lambda \overbrace{\underbrace{\sum_{(i,j)\in E} m_j\left(\frac{\pi_j}{m_j} - \frac{\pi_i}{n_i}\right)^2}_{\text{PageRank upper bound}}}^{\text{Node ranking}}}$$

Preserving Global Node Ranking

- Global node ranking preservation by minimizing the PageRank upper bound.
  - Express PageRank using the minimization of squared loss:
    $L = \sum_{j \in V} (\sum_{i \in P_j} \frac{\pi_i}{n_i} - \pi_j)^2$ ➔ infeasible for SGD
  - By Cauchy–Schwarz inequality, we have the upper bound:

$$\sum_{j \in V} \left(\sum_{i \in P_j} \frac{\pi_i}{n_i} - \pi_j \right)^2 \leq \sum_{(i,j) \in E} \underbrace{m_j \left(\frac{\pi_i}{n_i} - \frac{\pi_j}{m_j} \right)^2}_{\text{Proof: Sufficient condition for } L}.$$

PRUNE: Framework



- **Objective function**:

$$\arg\min_{\pi\geq 0, z\geq 0, W\geq 0} \overbrace{\sum_{(i,j)\in E}\left(\boldsymbol{z}_i^\top \boldsymbol{W}\boldsymbol{z}_j - \max\left\{0, \log\frac{M}{\alpha m_j n_i}\right\}\right)^2}^{\text{Proximity}} + \lambda \overbrace{\sum_{(i,j)\in E} m_j\left(\frac{\pi_j}{m_j} - \frac{\pi_i}{n_i}\right)^2}^{\text{Node ranking}}$$

$$\underbrace{\qquad\qquad\qquad}_{\text{Matrix tri-factorization}} \qquad \underbrace{\qquad\qquad\qquad}_{\text{PageRank upper bound}}$$

Four Key Criteria for Graph Embeddings: discussion on PRUNE

I. *Scalability*: uses only positive links and spends O(M Ω^2) time for each epoch (M: # of links, Ω: max number of neurons of a layer), Ω^2 << M

II. *Asymmetry*: (i, j) and (j, i) leads to different results (W and PageRank upper bound are asymmetric)

III. *Unity*: All objectives are jointly optimized under a multi-task Siamese neural network

IV. *Simplicity*: performs well with ***single*** hidden layer of **fewer and *consistent hyper-parameters*** across all datasets

Experiment Setups

- Datasets: three real-world networks
  - Webspam: web network
    - 114,529 webpages, 1,836,441 hyperlinks
  - Hep-Ph: paper citation network
    - 34,546 papers, 421,578 citations
  - FB Wall Post: social networks
    - 63,731 users, 831,401 wall post edges

Experiment Setups (cont.)

- Competitors:
  - DeepWalk, LINE, node2vec: popular for applications
  - SDNE: proposes another NN structure to embed networks
  - NRCL: one of the state-of-the-art network embedding models, specially designed for link prediction
- Model Setups:
  - 128-dim embedding and hidden layers, 64-dim proximity representation, ELU activation
  - Output layers: softplus for ranking scores and ReLU for proximity representation (avoid negative values)
  - Fix $\alpha = 5, \lambda = 0.01$

Task 1: Embedding for Learning-to-rank

- **Goal**: to rank non-spam nodes higher than spam, embeddings as node attributes.
- Linear SVM (C = 1.0) is used as learning-to-rank classifier: train on 80%, evaluate on 20% of datasets.
- Results: PRUNE (i.e., TriFac+Rank) >> TriFac (which only considers the proximity) >> Others

Table 2: Learning-to-rank performance (†: outperforms 2nd-best with p-value < 0.01).

Dataset	Evaluation	DeepWalk	LINE	node2vec	SDNE	NRCL	TriFac	PRUNE
Hep-Ph	Rank Corr.	0.485	0.430	0.494	0.353	0.327	0.554	**0.621**†
Webspam	AUC	0.821	0.818	0.843	0.800	0.839	0.821	**0.853**†
FB Wall Post	AUC	0.702	0.712	0.730	0.749	0.573	0.747	**0.765**†

Task 2: Classification and Regression

- Embedding are used for binary node classification on *Webspam* and *FB Wall Post* and node regression on *Hep-Ph* (citation prediction).
- Training on 80% nodes, predict labels of 20% nodes.
- Random Forest and Support Vector Regression are used for classification and regression, respectively.

Table 3: Classification and regression performance (†: outperforms 2nd-best with p-value < 0.01).

Dataset	Evaluation	DeepWalk	LINE	node2vec	SDNE	NRCL	TriFac	PRUNE
Hep-Ph	RMSE	12.079	12.307	11.909	12.451	12.429	11.967	**11.720**†
Webspam	AUC	0.620	0.597	0.622	0.605	0.578	0.576	**0.637**†
FB Wall Post	AUC	0.733	0.707	0.744	0.752	0.759	0.763	**0.775**†

Task 3: Link Prediction

- Setup: randomly split edges into 80%-20% train-test as positive + negative sampling.
- Logistic regression as the link prediction algorithm and embeddings learned on training set.
- Evaluated by AUC.

Table 4: Link prediction performance (†: outperforms 2nd-best with p-value < 0.01).

Dataset	DeepWalk	LINE	node2vec	SDNE	NRCL	TriFac	PRUNE
Hep-Ph	0.803	0.796	0.805	0.751	0.688	0.814	**0.861**†
Webspam	0.885	0.954	0.894	0.953	0.910	0.946	**0.973**†
FB Wall Post	0.828	0.781	0.853	0.855	0.731	0.858	**0.878**†

Conclusion

- Node Embedding is all about deciding 'what to preserve' and 'how to preserve'.
- Since it is unsupervised, parameter tuning is forbidden.
  - authors should provide a suggested parameter set for usage
- We argue (I) scalability, (II) asymmetry, (III) unity and (IV) simplicity as the key design criteria (anything else?)
  - Most of the existing models cannot satisfy all properties

Model	(I)	(II)	(III)	Model	(I)	(II)	(III)
Proximity Embedding [19]	✓	✓	✓	MMDW [22]			✓
SocDim [21]	✓		✓	SDNE [23]	✓		✓
Graph Factorization [1]	✓		✓	HOPE [15]	✓	✓	✓
DeepWalk [18]	✓		✓	node2vec [6]	✓		✓
TADW [28]		✓	✓	HSCA [30]		✓	✓
LINE [20]	✓	✓		LANE [8]			✓
GraRep [3]		✓		APP [31]	✓	✓	✓
DNGR [2]		✓	✓	M-NMF [24]			✓
TriDNR [17]	✓		✓	NRCL [26]	✓		✓
Our PRUNE	✓	✓	✓				

Graph Convolutional Networks

[1] Bruna, Zaremba, Szlam, and LeCun
"Spectral networks and locally connected networks on graphs."
ICLR 2014,

[2] Defferrard, Bresson, and Vandergheynst.
"Convolutional neural networks on graphs with fast localized spectral filtering."
NIPS 2016,

[3] Kipf and Welling.
"Semi-supervised classification with graph convolutional networks."
ICLR 2017.

Introduction

- Convolutional neural networks (CNNs) are good at extracting local stationary patterns.
- Recently, CNNs have dominated almost all the applications of image, video and speech.
- CNNs conduct filters to extract the latent patterns of locality for specific types of data.

- How about graphs (networks)?

Motivation (1/2)

- Graphs are common and informative topological structures to represent data. (Locality exists)

- The locality on *images*: the color of pixels are related to their two-dimensional neighborhood pixels.
- The locality on *vertices in a graph*: the type of nodes are related to their adjacency nodes in the graph.

- However, it is NOT straightforward to apply CNNs on on graphs.

Motivation (2/2)

- A simple strategy: conduct CNNs directly on the adjacency matrix A.
  - *Bad* performances compared to the conventional network-based methods.
  - Reason: The topological locality of graphs *cannot* be described in a 2-D matrix.
  - Consider $A' = PA$, where P is a permutation matrix, then A' describes the same graph as A, but is different in 2-D.
- Better strategy?
  - How to describe the locality in graphs?

Locality on Graphs (1/3)

- Main idea: Traditional CNNs summarize the local pattern using filters, and then reduce the size of *grids* via *pooling* and *subsampling* layers.
  - Why it works? The natural multi-scale clustering locality of grids! (e.g., pixels in images have "locality")
  - How about graphs?
  - Do graphs have multi-scale clustering locality?

- Remember that we *cannot* directly apply filters, pooling, subsampling on the adjacency matrix, so let's consider the topological view here.

Locality on Graphs (2/3)

Figure 1 illustrates a multiresolution clustering of a graph with the corresponding neighborhoods.

multi-scale clustering locality !

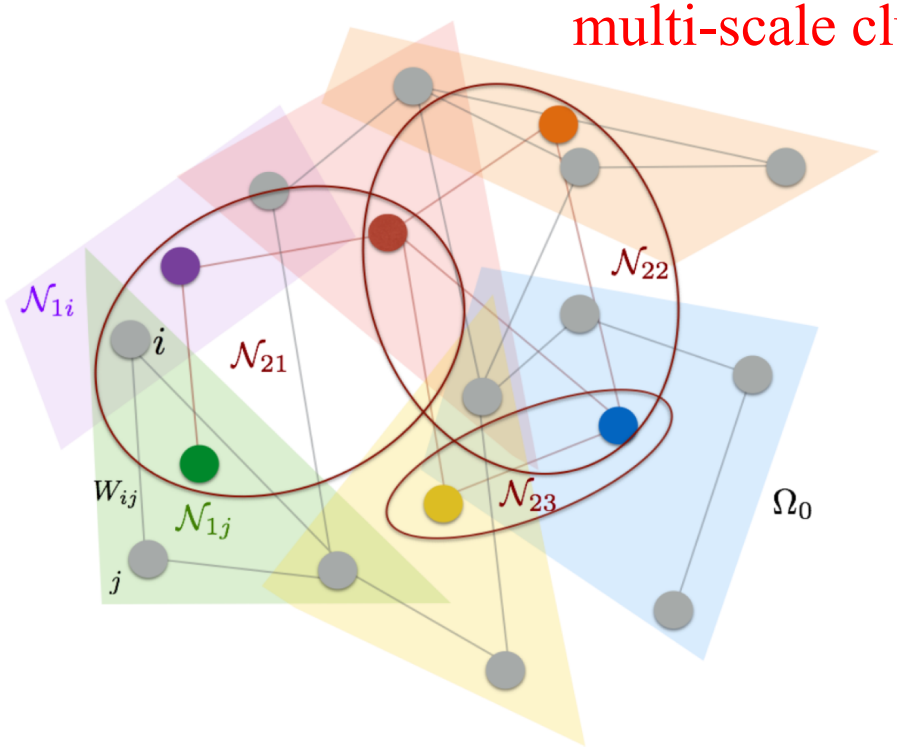


Figure 1: Undirected Graph $G = (\Omega_0, W)$ with two levels of clustering. The original points are drawn in gray.

Locality on Graphs (3/3)

- Another view of locality & multi-scale clustering: we can do parameter reduction!
  - i.e., use less parameters to represent the original entity.

- The first prototype of GCNs is established to perform *parameter reductions* on graphs
  - Bruna et al., "Spectral Networks and Deep Locally Connected Networks on Graphs" *ICLR* 2014
  - https://www.youtube.com/watch?v=xk17mfFxkag

Laplacian Graph Spectrum

- To capture the graph connectivity, we consider graph Laplacian matrices.

Commonly used in many applications (e.g., spectral clustering)

  - Laplacian Matrix: $L = D - W$
  - Symmetric normalized Laplacian: $L = I - D^{-1/2} W D^{-1/2}$
  - W: adjacency matrix, D: degree matrix
- As L is a real symmetric positive semi-definite matrix, it has a complete set of orthonormal eigenvectors $V = \{v_1, \ldots, v_N\}$.
- V is known as graph Fourier modes.
- We apply CNNs on these spectrums!

$$L^{\text{sym}}_{i,j} := \begin{cases} 1 & \text{if } i = j \text{ and } \deg(v_i) \neq 0 \\ -\frac{1}{\sqrt{\deg(v_i)\deg(v_j)}} & \text{if } i \neq j \text{ and } v_i \text{ is adjacent to } v_j \\ 0 & \text{otherwise.} \end{cases}$$

Constructing CNN Layers

- Let f_k be the dimension of representation vectors in the k-th layer, $k = 1, \ldots, M$.

$$\boxed{x_{k+1,j}} = h\left(V \boxed{\sum_{i=1}^{f_{k-1}}} F_{k,\boxed{i,j}} V^T \boxed{x_{k,i}}\right) \quad \boxed{(j = 1 \ldots f_k)}$$

- V is the eigen-matrix of the Laplacian matrix L.
- $F_{k,i,j}$ is a diagonal matrix.
  - Similar to the conv. Filter weights of traditional CNNs.
- Output embedding vector: $z_i := x_{M,i}$

GCN Training

- Supervised learning framework:

$$\min_{F} \quad \mathcal{L}(Z(W, X_0), Y)$$

- The *loss function L* differs from various tasks, for examples:
  - Regression (squared loss) $$\mathcal{L} = \sum_{i=1}^{N} \|z_i - y_i\|^2$$
  - Multi-class classification (cross-entropy)

$$\mathcal{L} = -\sum_{i=1}^{N} y_i \log z_i$$

Fast Localized Spectral Filters

- However, the computational cost of the "first prototype" is relatively high.
  - Several new papers attempt to speed it up.

- Defferrard et al. (2016) consider k-th order polynomial as spectral filters => k-localized
- Kipf et al. perform Chebyshev polynomial expansion for an approximation on the spectrums.
- (We omit the details here.)

Example: Node Classification (1/3)

- Kipf et al. (2017) considers a semi-supervised setting to demonstrate the ability of GCNs to capture the graph structure patterns.

- Input: a graph (network) with *extremely small amount* of known node labels.
- Output (Goal): predict the label of each node in the graph.

Example: Node Classification (2/3)

- A simple 2-layer GCN is conducted:

$$Z = f(X, A) = \mathrm{softmax}\Big(\hat{A}\ \underbrace{\mathrm{ReLU}\Big(\hat{A}XW^{(0)}\Big)}\ W^{(1)}\Big)$$

where $\hat{A} = \tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}$.

- The node feature (if available) can be encoded as X, or X can simply be one-hot encoding of nodes.
  - As the 0-th layers in the GCN.

Example: Node Classification (3/3)

- Loss function (Objective):

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

- Cross-entropy loss for multi-class classification.
- Optimize based on the nodes having labels.

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

- $W^{(0)}$ and $W^{(1)}$ are trained via SGD.

Performances

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

Table 2: Summary of results in terms of classification accuracy (in percent).

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

Current limitations

- Memory requirements
  - For large graphs, the Laplacian matrices and weight matrices are difficult to exactly store in the memory during the learning process.
- Directed edges
  - The current framework assume the affinity matrix (e.g., adjacency matrix) is symmetric => so that the Laplacian matrix is symmetric positive semi-definite.
- Time consuming for multi-layer learning
  - Similar to the memory issue, so current application usually use 2- or 3-layer GCNs (but can already obtain good results).

Source codes

- Defferrard et al. “Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering”, *NIPS* 2016.
  https://github.com/mdeff/cnn_graph

- Kipf et al. “Semi-Supervised Classification with Graph Convolutional Networks”, *ICLR* 2017.
  https://github.com/tkipf/gcn

Outline

- Introduction: What is Graph Embedding Learning?
- Node Embedding Methods
  - Matrix Factorization approaches
  - Random Walk approaches
  - Deep neural networks
- State-of-the-art Models
  - PRUNE
  - Graph convolutional networks
- Future Research Topics

Future research topics

- Graph embedding learning with meta-data of nodes
  - Additional images are available for nodes (e.g., users).
  - Corresponding raw-text descriptions are available for each node entity. (Homework 1)
  - …
  - Can additional information help?
- Better algorithms in terms of "the four key criteria for graph embeddings"
- Explainable graph embedding learning
- (and more)