Graph 4. Node Embedding

- Laplacian Eigenmaps (NIPS 2002)
- Random Walk (KDD 2014)

1

Lectures on Graph-based ML

- ✓ Graph 1-3. Social network analysis (e.g., HITS & PageRank)
- Graph 4. Node embedding (Laplacian eigenmaps vs. random walk methods)
- Graph 5-6. Graph neural networks (GCN, GAT, GIN, etc.)
- Graph 7-8. Knowledge graph embedding
- Graph 9-11. Neural solvers for combinatorial optimization (AR, NAR, LLM's)
- Graph 12-13. Graph-based learning for recommender systems (invited talks)
- Graph 14-15. Learning with heterogeneous graphs

2

Motivation for Graph Node Embedding

- GloVe co-occurrence graph → word embedding → NLP
- Hyperlinked websites → page embedding → websites classification
- Citation graphs → article embedding → literature classification
- Co-author graphs → author embedding → community detection
- Molecular structure graph → atom embedding → AI for science
- …

Unified View

- **Nodes** can be any objects (words, documents, authors, atoms, etc.)
- **Links** represent the interactions or dependencies among nodes.
- **Embedding Vectors**
  - Capturing the latent features of nodes based on graph structures
  - Supporting down-stream prediction tasks (node/graph classification, community detection, dense retrieval, etc.)

Node Embedding Methods

- Based on linked structures only (this lecture)
  - Matrix-factorization based methods (e.g., Laplacian Eigenmaps)
  - Random-walk based methods
- Based on both linked structures and node-specific features (next lecture)

Input and Output

- Input Graph $G = (V, A)$
  - V is the set of n nodes in the graph;
  - $A \in \mathbb{R}^{n \times n}$ is the adjacency matrix with $A_{ij} \geq 0$ and $A_{ii} = 0$ (zero at the diagonal).
  - Specifically, we focus on undirected graphs with $A_{ij} = A_{ji}$ (symmetric)
- Output Matrix $Z \in \mathbb{R}^{n \times d}$ $(d < n)$
  - Each row z_i is the embedding of a node;
  - Each column Z_j is a feature (latent factor) of the embedding space.

$Matrix\ Z$

$$\begin{array}{cc} & \\ z_i & \begin{bmatrix} z_{11} & z_{12} & \cdots & z_{1d} \\ z_{21} & z_{22} & \cdots & z_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ z_{n1} & z_{n2} & \cdots & z_{nd} \end{bmatrix} \\ & \quad\quad\quad Z_j \end{array}$$

Graph Laplacian Matrices

- Default Version

$$L \stackrel{\text{def}}{=} D - A \qquad \text{with } D \stackrel{\text{def}}{=} diag(D_{11}, \ldots, D_{nn}) \; and \; D_{ii} = \sum_j A_{ij}$$

- Symmetric Version

$$L_{sym} \stackrel{\text{def}}{=} D^{-\frac{1}{2}} L D^{-\frac{1}{2}} = \overbrace{D^{-\frac{1}{2}} D D^{-\frac{1}{2}}}^{I_{n \times n}} - \overbrace{D^{-\frac{1}{2}} A D^{-\frac{1}{2}}}^{A_{sym}} \qquad \left(A_{sym}[i,j] = \frac{A_{ij}}{\sqrt{D_{ii}}\sqrt{D_{jj}}} \right)$$

- Random-walk Version

$$L_{rw} \stackrel{\text{def}}{=} D^{-1} L = \overbrace{D^{-1} D}^{I_{n \times n}} - \overbrace{D^{-1} A}^{A_{rm}} \qquad \left(A_{rm}[i,j] = \frac{A_{ij}}{D_{ii}}, \;\; \sum_j A_{rm}[i,j] = 1 \right)$$

Properties of Graph Laplacian L

(Von Luxburg, 2007, https://arxiv.org/pdf/0711.0189.pdf)

- When A is symmetric, $L = \mathrm{D} - \mathrm{A}$ is also symmetric (obvious).
- L allows us to reinforce the smoothness of node embedding.
- L is positive semi-definite.
- L has n real-valued non-negative eigenvalues.
- λ_1 =0 is the smallest eigenvalue of L, and $\mathbf{1}_n$ as the corresponding eigenvector.
- L has a complete set of n eigenvectors (proof omitted).

Laplacian eigenmaps and spectral techniques for embedding and clustering [M. Belkin and P. Niyogi. NIPS 2002]

- **Task**: Given graph $G = (V, A)$, find the optimal solution

$$Z^* = arg \min_{Z \in \mathbb{R}^{n \times d}} \left| z_i - z_j \right|^2 A_{i,j} \quad \leftarrow \textit{Smoothness Penalty}$$

where the row vectors are node embeddings (d-dimensional vectors).

- **Optimal Solution**

$$Z^* = (u_2,\ u_3, \ldots, u_{d+1})$$

where $u_2,\ u_3, \ldots, u_{d+1}$ are the eigenvectors of the graph Laplacian (whichever the version of L, L_{sym} or L_{rm}) sorted by the eigenvalues $0 < \lambda_2 \le \lambda_3 \le \cdots \le \lambda_{k+1}$.

Smoothness Penalty (when $d = 1$)

- Let us consider $d = 1$ for now, i.e., $z_i, z_j \in \mathbb{R}$ (scalers) and $Z \in \mathbb{R}^{\mathrm{n}}$ (vector)

$$\begin{aligned}
\textstyle\sum_{i,j} \left| z_i - z_j \right|^2 A_{i,j} &= \textstyle\sum_{i,j} \left(z_i - z_j \right)^2 A_{ij} \\
&= \textstyle\sum_{i,j} \left(z_i^2 + z_j^2 - 2 z_i z_j \right) A_{ij} \\
&= \textstyle\sum_i z_i^2 \underbrace{\textstyle\sum_j A_{ij}}_{D_{ii}} + \textstyle\sum_j z_j^2 \underbrace{\textstyle\sum_i A_{ij}}_{D_{jj}} - \textstyle\sum_{ij} 2 z_i z_j A_{ij} \\
&= \boxed{2 \textstyle\sum_i z_i^2 D_{ii} - 2 \textstyle\sum_{ij} z_i z_j A_{ij}} = 2 Z^T L Z
\end{aligned}$$

- Also, we have

$$Z^T L Z = \textstyle\sum_{ij} z_i z_j L_{ij} = \textstyle\sum_{i,j} z_i z_j \left(D_{ij} - A_{ij} \right) = \boxed{\textstyle\sum_i z_i^2 D_{ii} - \textstyle\sum_{ij} z_i z_j A_{ij}}$$

Objective for Optimization (with $d = 1$)

$$Z^T L Z = \frac{1}{2}\sum_{i,j} \left|z_i - z_j\right|^2 A_{i,j} \geq 0 \qquad (Z \in \mathbb{R}^n)$$

$$\min_{Z\in\mathbb{R}^n} \sum_{i,j} \left|z_i - z_j\right|^2 A_{i,j} = \min_{Z\in\mathbb{R}^n} Z^T L Z$$

Smoothness Penalty

L is positive-semidefinite

($\because Z^T L Z \geq 0$ for any $Z \in \mathbb{R}^n$).

11

Being positive-semidefinite (PSD) of L …

1) For any $x \in \mathbb{R}^n$, we have $x^{\mathrm{T}} L x \geq 0$, according to the definition of a PSD matrix.
2) For any eigenvector $u_i \in \mathbb{R}^n$ of graph Laplacian L, we have $u_i^{\mathrm{T}} L u_i \geq 0$.
3) $L u_i = \lambda_i u_i \xrightarrow{multiplying\ u_i^T\ on\ both\ sides} u_i^T L u_i = \lambda_i \xrightarrow{imply} \lambda_i \geq 0, \forall i$ (by Item 2 above)
4) $\lambda_1 = 0$ is the smallest eigenvalue of L.
5) $u_1 = 1_n$ is the eigenvector corresponding to $\lambda_1 = 0$ (next slide).
6) Vector $1_n = Z^* = arg \min_{Z\in\mathbb{R}^n} Z^T L Z$ is naively optimal or 1D embedding.
7) We want to modify our objective as $Z^* = arg \min_{Z\perp u_1} Z^T L Z$, which leads to u_2 with $\lambda_2 > 0$.

12

Eigenvector $u_1 = 1_n$ with $\lambda_1 = 0$

- Starting with $L = D - A$, we have

$$L\mathbf{1}_n = D\mathbf{1}_n - A\mathbf{1}_n \qquad \text{(multiplying } \mathbf{1}_n \text{ on both sides)}$$

$$= \begin{pmatrix} D_{11} \\ D_{22} \\ \vdots \end{pmatrix} - \begin{pmatrix} \sum_{j=1}^{n} A_{1j} \\ \sum_{j=1}^{n} A_{2j} \\ \vdots \end{pmatrix}$$

$$= \begin{pmatrix} 0 \\ 0 \\ \vdots \end{pmatrix} = 0 \cdot \mathbf{1}_n$$

- Thus, we have $L\mathbf{1}_n = 0 \cdot \mathbf{1}_n$ which means $u_1 = \mathbf{1}_n$ and $\lambda_1 = 0$.

Smoothness penalty when $d > 1$

$$\sum_{i,j} \left|z_i - z_j\right|^2 A_{i,j} = \sum_{i,j} \sum_{k=1}^{d} \left(z_i^{(k)} - z_j^{(k)}\right)^2 A_{i,j} \qquad \left(z_i, z_j \in \mathbb{R}^d\right)$$

$$= \sum_{k=1}^{d} \sum_{i,j} \left(z_i^{(k)} - z_j^{(k)}\right)^2 A_{i,j}$$

$$= \sum_{k=1}^{d} 2{Z_k}^T L Z_k \qquad \text{(using the proof for } d = 1\text{)}$$

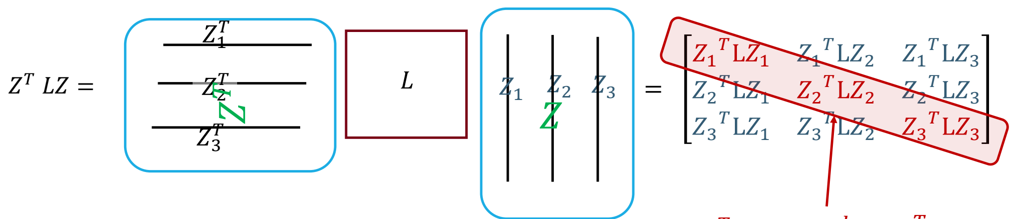
$$= 2tr(Z^T L Z) \qquad \text{(next slide)}$$

$$\Downarrow$$

$$\min_{Z \in \mathbb{R}^{n \times d}} \sum_{i,j} \left|z_i - z_j\right|^2 A_{i,j} = \min_{Z \in \mathbb{R}^{n \times d}} tr(Z^T L Z) \qquad (d \geq 1)$$

What is the trace of matrix $Z^T\ LZ$?

- Let $(Z_1, Z_2, \dots, Z_d)$ be the column vectors of matrix $Z \in \mathbb{R}^{n\times d}$.



$tr(Z^T\ LZ) = \sum_{k=1}^{d} Z_k{}^T LZ_k$

Optimal Embedding

- We want: $\boldsymbol{Z}^* = \underset{\boldsymbol{Z_k \perp 1_n}}{argmin} \sum_{k=1}^{d} Z_k{}^T LZ_k$
- When $d = 1$, $\boldsymbol{Z}^* = \underset{\boldsymbol{Z_1 \perp 1_n}}{argmin}\, Z_1{}^T LZ_1 = \boldsymbol{u}_2$
- When $d > 1$, for $k = 1$ to $d + 1$, find

  $\boldsymbol{Z}_{\boldsymbol{k+1}}^* = \underset{\boldsymbol{Z_{k+1}} \perp \{\boldsymbol{u_1}, \dots, \boldsymbol{u_k}\}}{argmin} Z_k{}^T LZ_k = \boldsymbol{u}_{k+1}$

  return $\boldsymbol{Z}^* = (\boldsymbol{u}_2, \dots, \boldsymbol{u}_{d+1})$

  where $\mathrm{tr}\left(Z^{*T} LZ^*\right) = \sum_{k=2}^{d+1} \boldsymbol{u}_k{}^T L\boldsymbol{u}_k = \lambda_2 + \lambda_3 + \dots + \lambda_{d+1}$
- How do we choose d?

Generalized Eigenvector Problem

[M. Belkin and P. Niyogi. NIPS 2002]

- **Standard eigenvector problem** is defined as to find all the vectors satisfying

$$L\boldsymbol{u} = \lambda\boldsymbol{u} \quad \text{or} \quad \boldsymbol{u}^T L\boldsymbol{u} = \lambda$$

  - Solution: eigenvectors $\boldsymbol{u}_1, \boldsymbol{u}_2, \dots \boldsymbol{u}_n$ ordered *by* $\lambda_1 \le \lambda_2 \le \dots \le \lambda_n$.
- **Generalized eigenvector problem** is defined as to find all the vectors satisfying

$$L\boldsymbol{y} = \lambda D\boldsymbol{y} \quad \text{or} \quad \boldsymbol{y}^T L\boldsymbol{y} = \lambda \quad (D \text{ is a diagonal matrix.})$$

  - Solution: D-orthonormal $\boldsymbol{y}_1, \boldsymbol{y}_2, \dots \boldsymbol{y}_n$ (when D is splitable)

$$\forall_{i,j} \boldsymbol{y}_i^T D \boldsymbol{y}_j = \begin{cases} 1 & if\ i = j \\ 0 & if\ i \ne j \end{cases} \quad \text{or} \quad Y^T DY = I,\ Y = (\boldsymbol{y}_1, \boldsymbol{y}_2, \dots \boldsymbol{y}_n)$$

Other matrix-factorization based methods

- **Laplacian Eigenmaps** (M. Belkin and P. Niyogi. NIPS 2002)

$$Z^* = arg\min_{Z_{n\times d}} \left|z_i - z_j\right|^2 A_{i,j} \quad .$$

- **Graph Factorization (**A. Ahmed et al., WWW 2013)

$$Z^* = arg\min_{Z_{n\times d}} \textstyle\sum_{i,j} |z_i^T z_j - A_{i,j}|$$

- **GraRep** (S. Cao et al., CIKM 2015)

$$\min_{orthononal Z^k} \textstyle\sum_{i,j} \left\| z_i^T z_j - \log\left(\frac{A_{i,j}^k}{\sum_l A_{l,j}^k}\right) + \log\frac{\lambda}{n} \right\|_2^2$$

Issue with the computational cost

- Eigen-decomposition of the full matrix is expensive when n is large.
- A chapter alternative is the random walk approach (next).

Node Embedding via Random Walk over a Graph
(e.g., DeepWalk by Perozzi et al., KDD 2014)

- Input graphs $G = (V, E)$, for example
  - **BlogCatalog**: a network of social relationships among blogger authors (10k)
  - **Flickr**: a network of the contacts between users (80k) on a photo sharing website
  - **YouTube**: a social network among users (1,139k) of a popular video sharing website
- Random walk for generating “word”/context pairs
  - From each node, randomly walk for *t* steps to obtain a sequence of nodes (“words”)
  - At each step, uniformly sample the next node from the neighbors of the current node
  - Apply a sliding window over the node sequences to obtain “word”/context pairs
  - Train a word2vec method (SkipGram) on the above training pairs to obtain the embedding of nodes

Other Variants of Random Walk Methods

- DeepWalk used a uniform sampling strategy, but other teleportation strategies are also possible (A Grover & J Leskovec, KDD 2016)
  - To reduce the probability of turning back to each node
  - To reduce the link weights for the nodes which have many links
  - To sample the context of each node via beam search (sampling multiple nodes per step instead sampling one node per step)
- DeepWalk uses SkipGram as the word embedding algorithm, but other choices (CBOW, GloVe, etc.) are also possible.

Concluding Remarks

- Matrix-factorization methods focus on the global smoothness of the entire graph (the eigen-decomposition of the graph Laplacian matrix).
- Random-walk methods focus on the local neighborhood of each node.
- The former could be too expensive for large graphs.
- The latter could be too simple for good performance.
- Both methods are task-agnostic, which is a limitation as the embeddings of nodes cannot be adapted to down-stream tasks.
- Both do not leverage node-specific features (even if available) as another weakness.
- Graph Neural Networks (GNNs) overcome both kinds of the limitations (next lecture).

References

- Von Luxburg, Ulrike. "A tutorial on spectral clustering." *Statistics and computing* 17.4 (2007).
- M. Belkin and P. Niyogi. “Laplacian eigenmaps and spectral techniques for embedding and clustering”, *NIPS 2002.*
- Perozzi, Bryan, Rami Al-Rfou, and Steven Skiena. "Deepwalk: Online learning of social representations." KDD 2014.
- A. Ahmed, N. Shervashidze, S. Narayanamurthy, V. Josifovski, A. Smola. Distribute Large-scale Natural Graph Factorization. WWW 2013.
- Cao, Shaosheng, Wei Lu, and Qiongkai Xu. "Grarep: Learning graph representations with global structural information." CIKM 2015.
- Hamilton, William L., Rex Ying, and Jure Leskovec. "Representation learning on graphs: Methods and applications." *arXiv preprint arXiv:1709.05584* (2017).
- Grover, Aditya, and Jure Leskovec. "node2vec: Scalable feature learning for networks." KDD 2016

Graph construction based on given node features
(M. Belkin and P. Niyogi. NIPS 2002)

- Let $x_1, x_2, \ldots, x_n \in \mathbb{R}^l$ be the given feature vectors of nodes, we can construct a graph based on
  - ε-neighborhoods ($\varepsilon > 0$): connect nodes i and j if $\|x_i - x_j\|^2 < \varepsilon$;
  - k-nearest neighbors ($k \in \mathbb{N}$): connect nodes i to j if i is among the k-nearest neighbors of j or if j is among the k-nearest neighbors of i.
- Choices of link weights in adjacency matrices
  - Simple-minded: $A_{ij} = 1$ if and only if nodes i and j are connected;
  - Heat kernel (with parameter $t > 0$): $A_{ij} = e^{-\frac{\|x_i - x_j\|^2}{t}}$ if nodes i and j are connected.