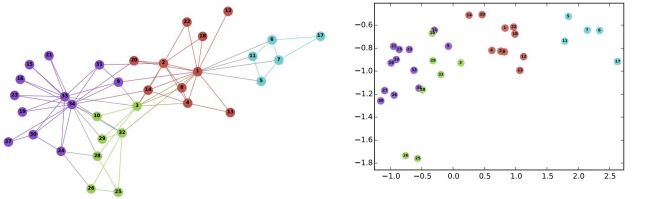
Network Embedding as Matrix Factorization: **Unifying** DeepWalk, LINE, PTE, and node2vec

Qiu et al. 2018

Presented by: Shadi Zabad

Background and Motivation

- Over the past decade, several algorithms have been proposed to learn **network embeddings**: Dense, latent representations that summarize the **structural properties of vertices in a network**.
  - DeepWalk (2014)
  - LINE (2015)
  - PTE (2015)
  - node2vec (2016)
- As we have seen in the case of DeepWalk, many of these algorithms are inspired by the **Skip Gram model** from NLP.



(a) Input: Karate Graph (b) Output: Representation

Credit: Perozzi et al. (2014)

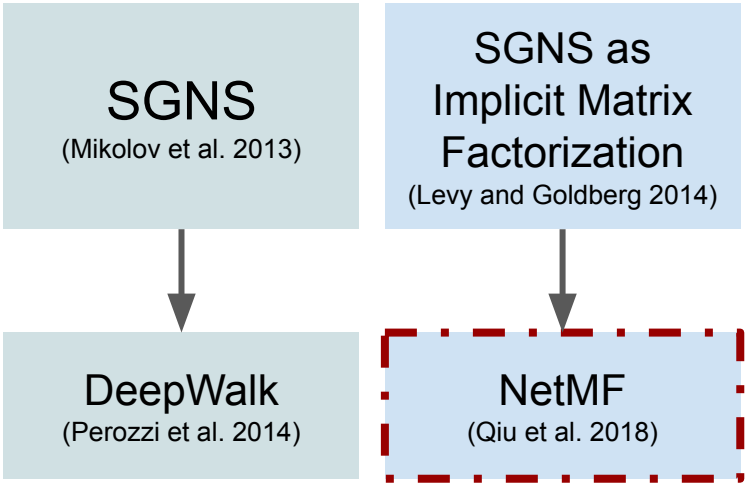
The Central Questions

Given the empirical success of these Graph Representation Learning algorithms, Qiu et al. set out to address the following questions:

1. What are the theoretical underpinnings of these different models?
2. Given this theoretical understanding, how are the algorithms related to each other (if at all)?
3. Can this unified theoretical model give us some hints as to how we can improve/augment them?

Tracing the Line of Discovery

- The **DeepWalk model** was inspired by the work of Mikolov et al. (2013) for learning word embeddings.
- In the same way, the paper by Qiu et al. is inspired by an earlier theoretical analysis by Levy and Goldberg (2014), in which they showed that the **Skip Gram Model with Negative Sampling (SGNS) can be formalized as a matrix factorization algorithm**.



Brief Overview of Levy and Goldberg's Analysis

- Recall that the Skip Gram Model aims to learn continuous embeddings for words **(w)** by modeling the strength of their association with their contexts **(c)** in a certain text corpus **(D)**.
- If we denote the probability of observing a **(w,c) pair** in a text corpus D as:

$$P(D = 1|w, c) = \sigma(\vec{w} \cdot \vec{c}) = \frac{1}{1 + e^{-\vec{w} \cdot \vec{c}}}$$

- Then we can frame the SGNS model as maximizing the following objective:

$$\log \sigma(\vec{w} \cdot \vec{c}) + k \cdot \mathbb{E}_{c_N \sim P_D} \left[\log \sigma(-\vec{w} \cdot \vec{c}_N)\right]$$

Sampling according to Unigram Distribution

Brief Overview of Levy and Goldberg's Analysis (2)

- This formulation of the SGNS model reveals that the model can be viewed as an (implicit) matrix factorization algorithm:

$$\underset{|V_W| \times d}{W} \cdot \underset{|V_C| \times d}{C^\top} = \underset{|V_W| \times |V_C|}{M}$$

- The Implicit Matrix (M) that the model is factorizing can, under certain mild assumptions, be shown to have the following form:

$$M_{ij}^{\text{SGNS}} = W_i \cdot C_j = \vec{w}_i \cdot \vec{c}_j = \log\left(\frac{\#(w,c) \cdot |D|}{\#(w) \cdot \#(c)}\right) - \log k$$

Concrete Question in the Graph Setting

- Given the insight from Levy and Goldberg's analysis, the concrete question for Qiu et al. then becomes:

What is the Implicit Matrix in the Graph Representation Learning Algorithms?

- As we will see, the answer to this question will differ depending on how each algorithm defines the (positive and negative) "context" in the graph setting.

DeepWalk

- Recall that DeepWalk generates its “corpus” by performing random walks of length L on the graph. We have to use our knowledge of random walks on graphs to fill in the terms in Levy and Goldberg’s equation:

$$\log\left(\frac{\#(w,c)\cdot|D|}{\#(w)\cdot\#(c)}\right)-\log k$$

- Our first task is to figure out how often **a pair of vertices (w, c)** can be found together in these paths within a window of size T.

DeepWalk (Task 1)

$$\log\left(\frac{\#(w,c)\cdot|D|}{\#(w)\cdot\#(c)}\right)-\log k$$

- This task can be broken into 2 steps:
  - (1) The probability of finding pairs (w,c) r steps away from each other:

$$\frac{d_w}{\mathrm{vol}(G)}(P^r)_{w,c} \qquad \frac{d_c}{\mathrm{vol}(G)}(P^r)_{c,w} \qquad P = D^{-1}A$$

  - (2) The probability of finding pairs (w, c) within any number of steps in a window of size T:

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

Note: $\mathrm{vol}(G)=\sum_{i=1}^{|V|}\sum_{j=1}^{|V|}A_{i,j}$

DeepWalk (Task 2)

$$\log\left(\frac{\#(w,c)\cdot|D|}{\#(w)\cdot\#(c)}\right)-\log k$$

- Task 2 is finding the number of times each vertex occurs in the corpus, and then computing the entire term inside the log(.).
- The probability of finding a vertex in a random path can be approximated by the stationary distribution:

$$\frac{d_c}{\mathrm{vol}(G)} \quad \frac{d_w}{\mathrm{vol}(G)}$$

- Putting everything together, we show that DeepWalk is implicitly factorizing a matrix that has the following entries for each pair of vertices:

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

The DeepWalk Implicit Matrix

- In matrix form, it can be shown that DeepWalk is factorizing the following matrix:

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

- The matrix highlighted in red can be decomposed into a product of symmetric matrices:

$$\left(\frac{1}{T}\sum_{r=1}^{T}\boldsymbol{P}^{r}\right)\boldsymbol{D}^{-1}=\left(\boldsymbol{D}^{-1/2}\right)\left(\boldsymbol{U}\left(\frac{1}{T}\sum_{r=1}^{T}\boldsymbol{\Lambda}^{r}\right)\boldsymbol{U}^{\top}\right)\left(\boldsymbol{D}^{-1/2}\right)$$

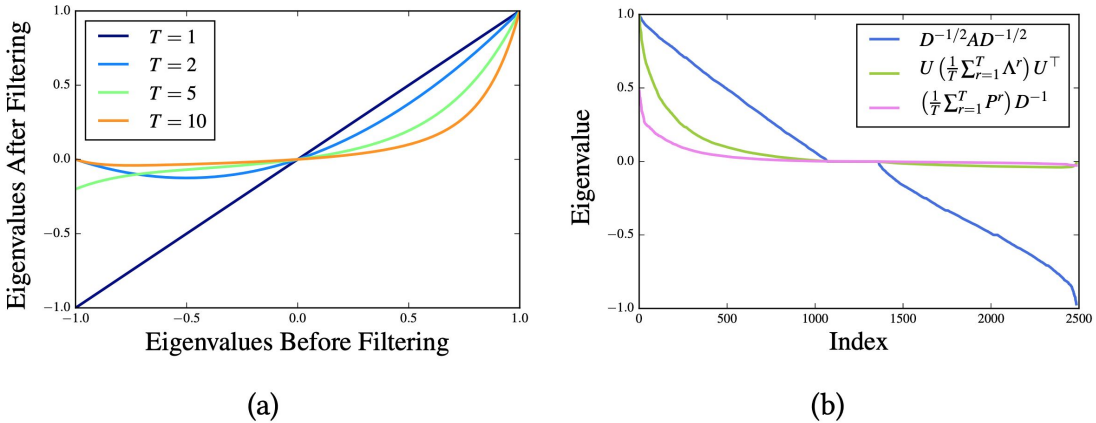
- **Very important insight**: DeepWalk is in effect applying a transformation to the eigenvalues of the normalized Laplacian of the network: $\boldsymbol{D}^{-1/2}\boldsymbol{A}\boldsymbol{D}^{-1/2}=\boldsymbol{I}-\boldsymbol{\mathcal{L}}$

Connection between DeepWalk and the Normalized Graph Laplacian

- The transformation that DeepWalk is applying to the Normalized Graph Laplacian is in effect a “filter”:

$$f(x) = \frac{1}{T} \sum_{r=1}^{T} x^r$$

- It encodes a preference for large positive eigenvalues that gets stronger with the window size T.



Proposed Algorithm: NetMF

- This analysis motivates a natural extension to the DeepWalk model: The Network Matrix Factorization Model (NetMF), where we factorize the following matrix:

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

- Where b and T and hyperparameters that we can tuned.
- Computing M for large values of T can be computationally challenging. Therefore, they approximate it with the top h eigenpairs of the normalized Laplacian:

$$\frac{\mathrm{vol}(G)}{b}\boldsymbol{D}^{-1/2}\boldsymbol{U}_h\left(\frac{1}{T}\sum_{r=1}^{T}\boldsymbol{\Lambda}_h^{r}\right)\boldsymbol{U}_h^{\top}\boldsymbol{D}^{-1/2}$$

Experimental Results: Performance Comparison

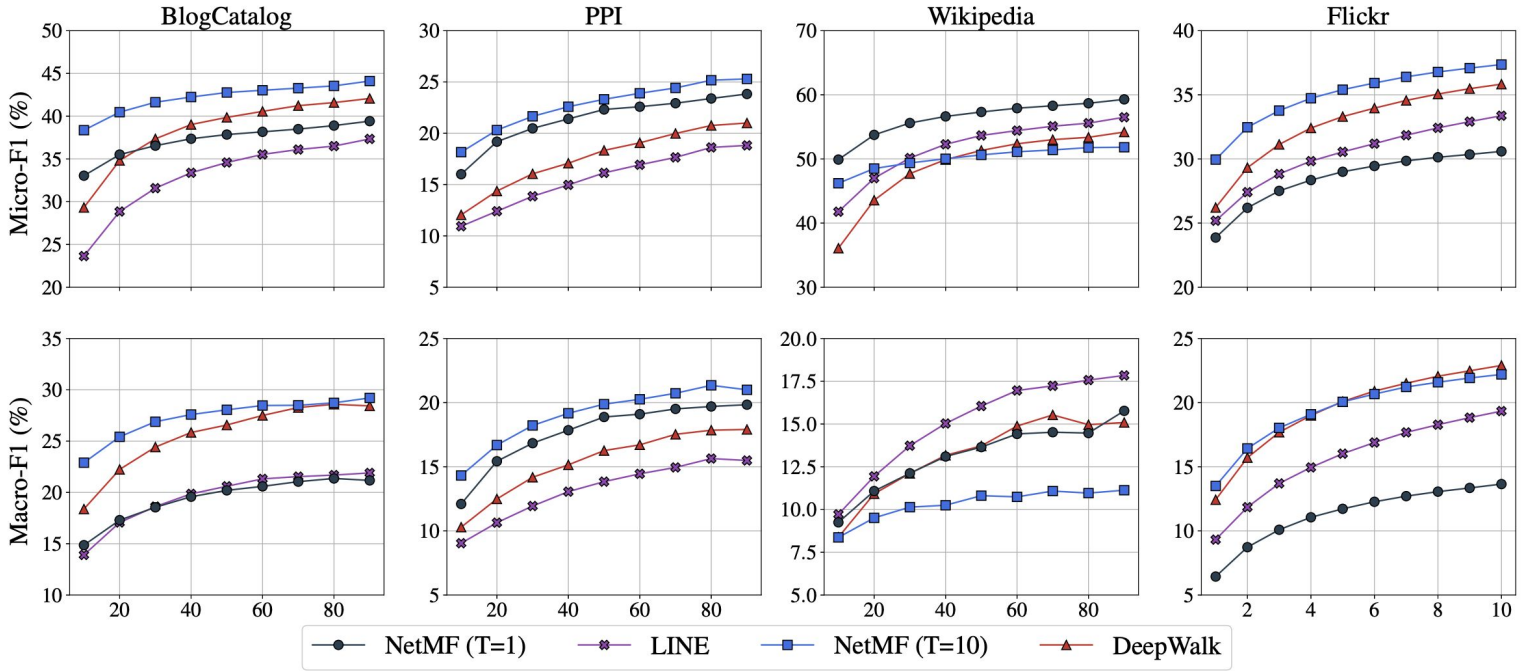


Figure 2: Predictive performance on varying the ratio of training data. The x-axis represents the ratio of labeled data (%), and the y-axis in the top and bottom rows denote the Micro-F1 and Macro-F1 scores respectively.

Limitations

- **Conceptual**: Levy and Goldberg showed that the matrix factorization framework performs as well as the SGNS model on some tasks but not on others. A similar issue could be present in the graph setting.
- **Computational**: An important advantage of the DeepWalk model is its scalability and parallelizability. It can even work with streaming data. The matrix factorization framework, on the other hand, faces some difficult computational challenges.

Future Directions

- Deriving implicit matrices for other algorithms that define clever ways to explore a node's structural properties.
- **Can we go the other way around?** By analyzing important theoretical properties of the Graph Laplacian, can we derive efficient, parallelizable algorithms for node embedding?
- Given recent advances in **tensor decomposition algorithms**, one question that may be worth pursuing is whether learning representations for multi-relational graphs can be formalized as a tensor decomposition algorithm.

Appendix

LINE and PTE

- LINE is a GRL model that works with (un-)weighted and (un-)directed graph structures. Its initial formulation is very similar to Levy and Goldberg's, where for a given pair of vertices (i,j) we maximize the objective:

$$\boldsymbol{A}_{i,j}\left(\log g\left(\boldsymbol{x}_i^\top \boldsymbol{y}_j\right) + b\mathbb{E}_{j'\sim P_N}\left[\log g\left(-\boldsymbol{x}_i^\top \boldsymbol{y}_{j'}\right)\right]\right)$$

- The implicit matrix for the LINE model depends on how we define the context (i.e. 1st vs. 2nd order proximity) and the way we sample negative contexts. If we assume that the sampling distribution is proportional to the out-degree of a vertex, then:

$$P_N(j) = d_j/\mathrm{vol}(G)$$

$$d_j = \sum_{k=1}^{|V|} \boldsymbol{A}_{j,k} \quad \mathrm{vol}(G) = \sum_{i=1}^{|V|}\sum_{j=1}^{|V|} \boldsymbol{A}_{i,j}$$

LINE and PTE (2)

- With these assumptions the authors show that the LINE model is implicitly factorizing the following matrix:

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

- **PTE (Predictive Text Embedding)** is an extension of the LINE model to graphs with multiple sub-networks (e.g. bipartite graphs). The authors show that it's implicitly factorizing:

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

Node2Vec

- Node2Vec performs a 2nd order random walk to generate its corpus D.
- The corpus in this case is formed of triplets of nodes.
- Given this corpus, the authors show that it's implicitly factorizing a matrix that has the following entries:

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

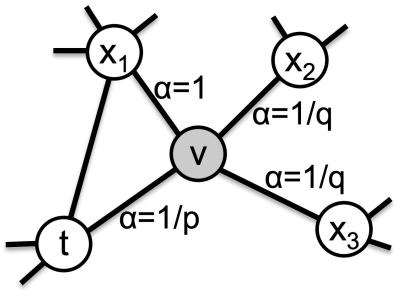


Figure 2: Illustration of the random walk procedure in *node2vec*. The walk just transitioned from t to v and is now evaluating its next step out of node v. Edge labels indicate search biases α.