NetSMF: Large-Scale Network Embedding as Sparse Matrix Factorization

Jiezhong Qiu

Tsinghua University

June 17, 2019

Joint work with Yuxiao Dong (MSR), Hao Ma (Facebook AI), Jian Li (IIIS, Tsinghua), Chi Wang (MSR), Kuansan Wang (MSR), and Jie Tang (DCST, Tsinghua)
Motivation and Problem Formulation

Problem Formulation
Give a network $G = (V, E)$, aim to learn a function $f : V \rightarrow \mathbb{R}^p$ to capture neighborhood similarity and community membership.

Applications:

- link prediction
- community detection
- label classification

Figure 1: A toy example (Figure from DeepWalk).
Two Genres of Network Embedding Algorithm

- **Local Context Methods:**
  - LINE, DeepWalk, node2vec, metapath2vec.
  - Usually be formulated as a skip-gram-like problem, and optimized by SGD.

- **Global Matrix Factorization Methods:**
  - NetMF, GraRep, HOPE.
  - Leverage global statistics of the input networks.
  - Not necessarily a gradient-based optimization problem.
  - Usually requires explicit construction of the matrix to be factorized.
Consider an undirected weighted graph $G = (V, E)$, where $|V| = n$ and $|E| = m$.

- **Adjacency matrix** $A \in \mathbb{R}^{n \times n}$:
  
  $A_{i,j} = \begin{cases} a_{i,j} > 0 & (i, j) \in E \\ 0 & (i, j) \notin E \end{cases}$.

- **Degree matrix** $D = \text{diag}(d_1, \cdots, d_n)$, where $d_i$ is the generalized degree of vertex $i$.

- **Volume of the graph** $G$: $\text{vol}(G) = \sum_i \sum_j A_{i,j}$.
Revisit DeepWalk and NetMF

NetSMF: Network Embedding as Sparse Matrix Factorization

Experimental Results
DeepWalk and NetMF

Input
\( G=(V,E) \)

Random Walk

Skip-gram

Output: Node Embedding
DeepWalk and NetMF

Input
\[ G = (V, E) \]

Random Walk

Skip-gram

Output:
Node Embedding

Levy & Goldberg (NIPS 14)

\[
\log \left( \frac{\#(w, c) \cdot |D|}{b \#(w) \#(c)} \right)
\]

- \#(w, c): Co-occurrence of \( w \) and \( c \)
- \(|D|\): Total number of word-context pairs
- \#(w): Occurrence of word \( w \)
- \#(c): Occurrence of context \( c \)
- \( b \): Number of negative samples
DeepWalk and NetMF

Input: \( G=(V,E) \)

Random Walk

\[ \log \left( \frac{\#(w,c) \cdot |D|}{b \#(w) \#(c)} \right) \]

- \( \#(w,c) \): Co-occurrence of w and c
- \( |D| \): Total number of word-context pairs
- \( b \): Number of negative samples
- \( \#(w) \): Occurrence of word w
- \( \#(c) \): Occurrence of context c

Output: Node Embedding

Levy & Goldberg (NIPS 14)

\[ \text{Input, projection, output} \]

Skip-gram

\[ w(0), w(1), w(2) \]
DeepWalk and NetMF

Input:
\[ G = (V, E) \]

Random Walk

Skip-gram

Output: Node Embedding

\[
\log \left( \frac{\text{vol}(G)}{b} \left( \frac{1}{T} \sum_{r=1}^{T} (D^{-1} A)^r \right) D^{-1} \right)
\]

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

- **A**: Adjacency matrix
- **D**: Degree matrix
- \( \text{vol}(G) = \sum_i \sum_j A_{i,j} \)
- \( b \): Number of negative samples

Levy & Goldberg (NIPS 14)
DeepWalk and NetMF

Input
\( G=(V,E) \)

\[ \log \left( \frac{\text{vol}(G)}{b} \left( \frac{1}{T} \sum_{r=1}^{T} (D^{-1}A)^r \right) D^{-1} \right) \]

Matrix Factorization

Random Walk

......

Skip-gram

Output: Node Embedding

Levy & Goldberg (NIPS 14)

\[ \log \left( \frac{\#(w,c)}{b\#(w)\#(c)} \right) \]
Revisit DeepWalk and NetMF

NetSMF: Network Embedding as Sparse Matrix Factorization

Experimental Results
Contents

Revisit DeepWalk and NetMF

NetSMF: Network Embedding as Sparse Matrix Factorization

Experimental Results
For small world networks, 

\[
\frac{\text{vol}(G)}{b} \left( \frac{1}{T} \sum_{r=1}^{T} \left( D^{-1} A \right)^{r} \right) D^{-1} \text{ is always a dense matrix}.
\]
Computation Challenges of NetMF

For small world networks,

\[
\frac{\text{vol}(G)}{b} \left( \frac{1}{T} \sum_{r=1}^{T} (D^{-1}A)^r \right) D^{-1} \text{ is always a dense matrix}.
\]

Why?

▶ In small world networks, each pair of vertices \((i, j)\) can reach each other in a small number of hops.
▶ Make the corresponding matrix entry a positive value.
For small world networks,
\[
\frac{\text{vol}(G)}{b} \left( \frac{1}{T} \sum_{r=1}^{T} (D^{-1}A)^r \right) D^{-1} \text{ is always a dense matrix .}
\]

Why?

- In small world networks, each pair of vertices \((i,j)\) can reach each other in a small number of hops.
- Make the corresponding matrix entry a positive value.

Idea

- Sparse matrix is easier to handle.
- Can we achieve a matrix sparse but ‘good enough’ matrix.
Observation

Definition
For $\sum_{r=1}^{T} \alpha_r = 1$ and $\alpha_r$ non-negative,

$$L = D - \sum_{r=1}^{T} \alpha_r D \left( D^{-1} A \right)^r$$

is a $T$-degree random-walk matrix polynomial.

Observation
For $\alpha_1 = \cdots = \alpha_T = \frac{1}{T}$:

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

$$= \log^\circ \left( \frac{\text{vol}(G)}{b} D^{-1} (D - L) D^{-1} \right)$$

$$\approx \log^\circ \left( \frac{\text{vol}(G)}{b} D^{-1} (D - \tilde{L}) D^{-1} \right)$$
Theorem

[CCL+ 15] For random-walk matrix polynomial

$$L = D - \sum_{r=1}^{T} \alpha_r D \left( D^{-1} A \right)^r,$$

one can construct, in time

$$O(T^2 m \epsilon^{-2} \log^2 n),$$

a $$(1 + \epsilon)$$-spectral sparsifier, $$\tilde{L}$$, with

$$O(n \log n \epsilon^{-2})$$ non-zeros. For unweighted graphs, the time complexity can be reduced to

$$O(T^2 m \epsilon^{-2} \log n).$$
The proposed NetSMF algorithm consists of three steps:

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

$$\text{trunc}_\log^o \left( \frac{\text{vol}(G)}{b} D^{-1} (D - \tilde{L}) D^{-1} \right)$$

- Truncated randomized singular value decomposition.
Algorithm Details

PathSampling:
- Sample an edge \((u, v)\) from edge set.
- Start very short random walk from \(u\) and arrive \(u'\).
- Start very short random walk from \(v\) and arrive \(v'\).
- Record vertex pair \((u', v')\).

Randomized SVD:
- Project origin matrix to low dimensional space by Gaussian random matrix.
- Deal with the projected small matrix.
Figure 2: The System Design of NetSMF.
Contents

Revisit DeepWalk and NetMF

NetSMF: Network Embedding as Sparse Matrix Factorization

Experimental Results
Setup

Label Classification:
- BlogCatalog, PPI, Flickr, YouTube, OAG.
- Logistic Regression
- NetSMF ($T = 10$), NetMF ($T = 10$), DeepWalk, LINE.

Table 1: Statistics of Datasets.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>BlogCatalog</th>
<th>PPI</th>
<th>Flickr</th>
<th>YouTube</th>
<th>OAG</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>V</td>
<td>$</td>
<td>10,312</td>
<td>3,890</td>
<td>80,513</td>
</tr>
<tr>
<td>$</td>
<td>E</td>
<td>$</td>
<td>333,983</td>
<td>76,584</td>
<td>5,899,882</td>
</tr>
<tr>
<td>#Labels</td>
<td>39</td>
<td>50</td>
<td>195</td>
<td>47</td>
<td>19</td>
</tr>
</tbody>
</table>
Experimental Results

Figure 3: 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.
### Table 2: Running Time

<table>
<thead>
<tr>
<th>Data Set</th>
<th>LINE</th>
<th>DeepWalk</th>
<th>node2vec</th>
<th>NetMF</th>
<th>NetSMF</th>
</tr>
</thead>
<tbody>
<tr>
<td>BlogCatalog</td>
<td>40 mins</td>
<td>12 mins</td>
<td>56 mins</td>
<td>2 mins</td>
<td>13 mins</td>
</tr>
<tr>
<td>PPI</td>
<td>41 mins</td>
<td>4 mins</td>
<td>4 mins</td>
<td>16 secs</td>
<td>10 secs</td>
</tr>
<tr>
<td>Flickr</td>
<td>42 mins</td>
<td>2.2 hours</td>
<td>21 hours</td>
<td>2 hours</td>
<td>48 mins</td>
</tr>
<tr>
<td>YouTube</td>
<td>46 mins</td>
<td>1 day</td>
<td>4 days</td>
<td>×</td>
<td>4.1 hours</td>
</tr>
<tr>
<td>OAG</td>
<td>2.6 hours</td>
<td>–</td>
<td>–</td>
<td>×</td>
<td>24 hours</td>
</tr>
</tbody>
</table>
Conclusion and Future Work

We propose NetSMF, a scalable, efficient, and effective network embedding algorithm.

Future Work

- A distributed-memory implementation.
- Extension to directed, dynamic, heterogeneous graphs.
Thanks.

- Network Embedding as Matrix Factorization: Unifying DeepWalk, LINE, PTE, and node2vec (WSDM ’18)
- NetSMF: Network Embedding as Sparse Matrix Factorization (WebConf ’19)

Code for NetMF available at github.com/xptree/NetMF
Code for NetSMF available at github.com/xptree/NetSMF

Q&A
Recall the objective of skip-gram model:

$$\min_{X, Y} \mathcal{L}(X, Y)$$

where

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

**Theorem**

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

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

$$\frac{\#(w)}{|D|} \xrightarrow{p} \frac{d_w}{\text{vol}(G)} \quad \text{and} \quad \frac{\#(c)}{|D|} \xrightarrow{p} \frac{d_c}{\text{vol}(G)}.$$
Denote $M = D^{-1} (D - L) D^{-1}$ in

$$\text{trunc}_{\log^o} \left( \frac{\text{vol}(G)}{b} D^{-1} (D - \tilde{L}) D^{-1} \right),$$

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

**Theorem**

The singular value of $\tilde{M} - M$ satisfies

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

**Theorem**

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

$$\left\| \text{trunc}_{\log^o} \left( \frac{\text{vol}(G)}{b} \tilde{M} \right) - \text{trunc}_{\log^o} \left( \frac{\text{vol}(G)}{b} M \right) \right\|_F \leq \frac{4\epsilon \text{vol}(G)}{b \sqrt{d_{\min}}} \sqrt{\sum_{i=1}^{n} \frac{1}{d_i}}.$$
Spectrally Similar

Definition
Suppose $G = (V, E, A)$ and $\tilde{G} = (V, \tilde{E}, \tilde{A})$ are two weighted undirected networks. Let $L = D_G - A$ and $\tilde{L} = D_{\tilde{G}} - \tilde{A}$ be their Laplacian matrices, respectively. We define $G$ and $\tilde{G}$ are $(1 + \epsilon)$-spectrally similar if

$$\forall x \in \mathbb{R}^n, (1 - \epsilon) \cdot x^T \tilde{L}x \leq x^T Lx \leq (1 + \epsilon) \cdot x^T \tilde{L}x.$$
Algorithm 1: NetSMF

Input: A social network $G = (V, E, A)$ which we want to learn network embedding;
       The number of non-zeros $M$ in the sparsifier; The dimension of embedding $d$.

Output: An embedding matrix of size $n \times d$, each row corresponding to a vertex.

1. $\tilde{G} \leftarrow (V, \emptyset, \tilde{A} = 0)$  /* Create an empty network with $E = \emptyset$ and $\tilde{A} = 0$. */
2. for $i \leftarrow 1$ to $M$ do
3.   Uniformly pick an edge $e = (u, v) \in E$
4.   Uniformly pick an integer $r \in [T]$
5.   $u', v', Z \leftarrow \text{PathSampling}(e, r)$
6.   Add an edge $(u', v', \frac{2rm}{MZ})$ to $\tilde{G}$  /* Parallel edges will be merged into one edge, with their weights summed up together. */
7. end
8. Compute $\tilde{L}$ to be the unnormalized graph Laplacian of $\tilde{G}$
9. Compute $\tilde{M} = D^{-1} \left( D - \tilde{L} \right) D^{-1}$
10. $U_d, \Sigma_d, V_d \leftarrow \text{RandomizedSVD}(\text{trunc\_log}^\circ \left( \frac{\text{vol}(G)}{b} \tilde{M} \right), d)$
11. return $U_d\sqrt{\Sigma_d}$ as network embeddings
Algorithm 2: PathSampling algorithm as described in [CCL+15].

1. Procedure PathSampling\((e = (u, v), r)\)
2. Uniformly pick an integer \(k \in [r]\)
3. Perform \((k - 1)\)-step random walk from \(u\) to \(u_0\)
4. Perform \((r - k)\)-step random walk from \(v\) to \(u_r\)
5. Keep track of \(Z(p) = \sum_{i=1}^{r} \frac{2}{A_{u_{i-1}, u_i}}\) along the length-\(r\) path \(p\) between \(u_0\) and \(u_r\)
6. return \(u_0, u_r, Z(p)\)
Algorithm 3: Randomized SVD on NetMF Matrix Sparsifier

1. Procedure RandomizedSVD($A$, $d$)
2. Sampling Gaussian random matrix $O$ \hspace{1cm} // $O \in \mathbb{R}^{n \times d}$
3. Compute sample matrix $Y = A^T O = AO$ \hspace{1cm} // $Y \in \mathbb{R}^{n \times d}$
4. Orthonormalize $Y$
5. Compute $B = AY$ \hspace{1cm} // $B \in \mathbb{R}^{n \times d}$
6. Sample another Gaussian random matrix $P$ \hspace{1cm} // $P \in \mathbb{R}^{d \times d}$
7. Compute sample matrix of $Z = BP$ \hspace{1cm} // $Z \in \mathbb{R}^{n \times d}$
8. Orthonormalize $Z$
9. Compute $C = Z^T B$ \hspace{1cm} // $C \in \mathbb{R}^{d \times d}$
10. Run Jacobi SVD on $C = U \Sigma V^T$
11. return $ZU$, $\Sigma$, $YV$

/* Result matrices are of shape $n \times d$, $d \times d$, $n \times d$ resp. */
### Table 3: Time and Space Complexity of NetSMF.

<table>
<thead>
<tr>
<th>Step</th>
<th>Time</th>
<th>Space</th>
</tr>
</thead>
</table>
| **Step 1** | $O(MT \log n)$ for weighted networks  
                                      $O(MT)$ for unweighted networks | $O(M + n + m)$ |
| **Step 2** | $O(M)$                                   | $O(M + n)$ |
| **Step 3** | $O(Md + nd^2 + d^3)$                  | $O(M + nd)$ |