Fast Graph Learning with Unique Optimal Solutions

Sami Abu-El-Haija 1 Valentino Crespi 1 Greg Ver Steeg 1 Aram Galstyan 1

Abstract
Graph Representation Learning (GRL) has been advancing at an unprecedented rate. However, many results rely on careful design and tuning of architectures, objectives, and training schemes. We propose efficient GRL methods that optimize convexified objectives with known closed form solutions. Guaranteed convergence to a global optimum releases practitioners from hyper-parameter and architecture tuning. Nevertheless, our proposed method achieves competitive or state-of-the-art performance on popular GRL tasks while providing orders of magnitude speedup. Although the design matrix (M) of our objective is expensive to compute, we exploit results from random matrix theory to approximate solutions in linear time while avoiding an explicit calculation of M.

Our code is online:  http://github.com/samihaija/tf-fsvd

1. Introduction
Nowadays, many advancements in graph representation learning (GRL) are supported by powerful computational frameworks such as TensorFlow (Abadi et al., 2016) and PyTorch (Paszke et al., 2019). Implementing popular transformations, such as neural layers for composing deeper networks, these frameworks facilitate creative and theoretically-justified models (Kipf & Welling, 2017; Hamilton et al., 2017; Veličković et al., 2018; Qiu et al., 2018; Xu et al., 2019; Abu-El-Haija et al., 2019; Chen et al., 2020). Unfortunately, many of these methods contain hyperparameters that need to be tuned (such as learning rate, regularization coefficient, depth and width of the network), and training takes a long time (e.g. minutes) even on smaller datasets. Further, it is rather uncommon to check for convergence, e.g., by checking $\nabla W J = 0$ where $J$ is mean training error.

To address these weaknesses, our goals are: (i) to quickly train (ii) competitive GRL models by posing convex objectives and estimating optimal solutions in closed-form, hence (iii) relieving practitioners from hyperparameter tuning or convergence checks. Our goals remind us of a classical learning technique that has been used for decades.

Specifically, Singular Value Decomposition (SVD) aided many advancements including in natural language processing (NLP), e.g., Latent Semantic Indexing, (LSI, Deerwester et al., 1990), matrix completion, e.g., for recommender systems (Koren et al., 2009); and computing matrix properties, such as the pseudoinverse (Golub & Loan, 1996). Beyond its wide applicability, benefits of SVD include simple training routines (usually, a few code lines performs learning), speed of training (fast SVD algorithms are well-known), fewer hyperparameters such as the lack of learning rate, regularization coefficient, model architectures, etc.

SVD periodically re-appears within powerful yet simple methods, competing on state-of-the-art. The common practice is to design a matrix M, such that its decomposition (e.g., via SVD), provides an estimate for learning a model given an objective. For instance, Levy & Goldberg (2014) show that the learning of NLP skipgram models such as word2vec (Mikolov et al., 2013) and GloVe (Pennington et al., 2014), can be approximated by the SVD of a Shifted Positive Pointwise Mutual Information matrix.

In GRL, Chen et al. (2017); Qiu et al. (2018); Abu-El-Haija et al. (2018) have approximated methods of DeepWalk (Perozzi et al., 2014) and Node2Vec (Grover & Leskovec, 2016) via decomposition of some matrix M. However, their decomposition requires M to be either (a) exactly calculated or (b) sampled entry-wise, but (a) is unnecessarily expensive for real-world large networks (due to Small World Phenomenon, §2.2) and (b) incurs unnecessary estimation errors. On the other hand, known algorithms in matrix theory can decompose any matrix M without explicitly knowing M. Specifically, it is sufficient to provide a function $f_M(\cdot) = \langle M, \cdot \rangle$ that can multiply M with arbitrary vectors (§4). We argue that if the popular frameworks (e.g., TensorFlow) implement a functional SVD, that accept $f_M(\cdot)$ rather than M, then modern practitioners may find it useful.

Although a functional SVD could assist various fields, we focus our interest on GRL models, specifically ones that can output predictions per node (e.g., for node classification) or a per edge (e.g., for link prediction). GRL can benefit from a functional SVD, as many methods can be approximated by

---

1 USC Information Sciences Institute. Correspondence to: {haija, vcrespi, gregv, galstyan}@isi.edu.
SVD of some matrix $M = \sum_i g_i(A)^i$, where adjacency $A$ and its transformations $g_i$ are all sparse matrices, though the power $i \in \mathbb{N}_+$ likely yields dense $g_i(A)^i$ for $i \geq 6$ hence requiring quadratic storage ($\S 2.2$). In these cases, even though $M$ is expensive to calculate, it is much cheaper to multiply it by vectors, therefore performing learning using memory and computation complexity that is linear in network size, while sacrificing no more error than SVD on dense $M$.

We review powerful GRL methods ($\S 2.3$) that we convexify ($\S 3$) by following two steps: (i) linearize their model and (ii) replace their objective with Frobenius norm minimization. Such convexification elicits (randomized) SVD as a natural choice for obtaining (approximate) optimum solutions.

We now summarize our contributions:

1. We open-source a functional SVD ($\S 4$) implementation of the randomized algorithm of Halko et al. (2009).
2. We approximate embedding and message passing methods via SVD ($\S 3$), showing competitive performance with state-of-the-art, yet much faster to train ($\S 5$).
3. Our analysis proves that our learning is fast and our estimation error can be made arbitrarily small ($\S 4.3$).

2. Preliminaries

2.1. Truncated Singular Value Decomposition (SVD)

Truncated (top-$k$) Singular Value Decomposition (SVD) estimates input matrix $M \in \mathbb{R}^{r \times c}$ with low-rank $\tilde{M}$ that minimizes the Frobenius norm of the error:

$$\min_{\tilde{M}} ||M - \tilde{M}||_F^2, \text{ subject to: } \text{rank}(\tilde{M}) \leq k, \quad (1)$$

while parameterizing $\tilde{M}$ as $\tilde{M} = U_k S_k V_k^\top$, subject to, $S_k \in \mathbb{R}^{k \times k}$ being diagonal matrix and columns of $U_k, V_k$ being orthonormal. It turns out, the minimizer of Frobenius norm $||.||_F$ recovers the top-$k$ singular values on the diagonal of $S_k$, with their corresponding left- and right-singular vectors, respectively, on the columns of unitary matrices $U_k \in \mathbb{R}^{r \times k}$ and $V_k \in \mathbb{R}^{c \times k}$ (a.k.a, the singular bases).

SVD has many applications, two of which are used in our work: (1) it has been used for embedding and matrix completion; and (2) it can estimate the pseudoinverse of matrix $\tilde{M}$ (a.k.a., Moore-Penrose inverse), as:

$$M^\dagger = M^\top (MM^\top)^{-1} \approx V_k S_k^{-1} U_k^\top, \quad (2)$$

where one calculates inverse $S^{-1}$ by reciprocating entries of diagonal matrix $S$. The $\approx$ becomes $= \text{when } k \geq \text{rank}(M)$, due to (Eckart & Young, 1936; Golub & Loan, 1996).

2.2. Small World Phenomenon (Karlinthy, 1929)

Karlinthy (1929) hypothesized that the world is smaller than we think. Specifically, any person is at most 5 links apart from any other person. For instance, you know someone [that knows someone] $\times 4$ that knows Joe Biden, as well as many other influential figures! This was experimentally supported by Travers & Milgram (1969), when individuals were recruited (through advertisements) in Kansas and Nebraska to deliver a letter to a person in Massachusetts, where each person can forward the letter to one of their direct acquaintances. For the people who completed the study, the average path length was below 6. Further, Backstrom et al. (2012) analyze the Facebook network of $\approx 700$ million active users (at the time) and find that average distance between any two individuals is less than 5.

Network of $n$ nodes and $m$ edges can be represented with an $n \times n$ adjacency matrix $A$ with $O(m)$ nonzero entries. $A$ can be efficiently stored as a sparse matrix since $m \ll n^2$ for most realistic graphs. If nodes $u$ and $v$ are directly connected (1 step away), then entry $A_{uv}$ will be nonzero. If they are 2 steps away, then $(A^2)_{uv}$ will be nonzero, etc. The Small World Phenomenon implies $A^i$ is dense for $i \geq 6$ with $O(n^2)$ nonzeros, as most node pairs are within 6 hops.

2.3. Graph Representation Learning (GRL)

We review two popular GRL methods and tasks: (i) message passing for node classification and (ii) network embedding for link prediction, as they cover many practical applications including in recommender systems, biological e.g. protein-protein interactions, social and citation networks:

(i) Many message passing models can be written as:

$$H = \sigma_L \left( g_L(A) \ldots \sigma_2 \left( g_2(A) \sigma_1 \left( g_1(A) X W_1 \right) \right) W_2 \ldots W_L \right) \quad (3)$$

where $L$ is the number of layers, matrix $X \in \mathbb{R}^{n \times d}$ contains $d$ features per node, $W$’s are trainable parameters, $\sigma$ denote activations (e.g. ReLu), and $g$ is some (possibly trainable) transformation of adjacency matrix. GCN (Kipf & Welling, 2017) set $g$ to symmetric normalization per renormalization trick, GAT (Velicković et al., 2018) set $g(A) = A \circ \text{MultiHeadedAttention and GIN (Xu et al., 2019)}$ as $g(A) = A + (1 + \epsilon) I$ with identity $I$ and $\epsilon > 0$. For node classification, it is common to set $\sigma_L ^{(i)} = \text{softmax (applied row-wise)},$ specify the size of $W_L$ s.t $H \in \mathbb{R}^{n \times y}$ where $y$ is number of classes, and optimize cross-entropy objective:

$$\min_{(W_i)_{i=1}} - y \circ \log(H) - (1 - y) \circ \log(1 - H), \quad (4)$$

where $Y$ is a binary matrix with one-hot rows indicating node labels. $\circ$ is Hadamard product. In semi-supervised node classification settings where not all nodes are labeled, before measuring the objective, subset of rows can be kept in $Y$ and $H$ that correspond to labeled nodes.
(ii) **Network embedding** methods map nodes onto a \( z \)-dimensional vector space \( Z \in \mathbb{R}^{n \times z} \). Classical approaches compute \( Z \) by directly decomposing a transformation of the adjacency matrix, such as the graph laplacian \( (\text{Belkin \\& Niyogi, } 2003) \). The embeddings can then be used to solve graph tasks, such as link prediction or node classifications. Modern approaches train skipgram models (e.g. word2vec \( (\text{Mikolov et al., } 2013) \), GloVe \( (\text{Pennington et al., } 2014) \)) on sampled random walks. It has been shown that these skipgram network embedding methods, including DeepWalk \( (\text{Perozzi et al., } 2014) \) and node2vec \( (\text{Grover \\& Leskovec, } 2016) \), with a learning process of walk sampling followed by positional embedding, can be approximated as a matrix decomposition \( (\text{Chen et al., } 2017; \text{Abu-El-Haija et al., } 2018; \text{Qiu et al., } 2018) \). We point the curious reader to the listed papers for how the decomposition was derived, but show here the decomposition derived in \( (\text{WYS, Abu-El-Haija et al., } 2018) \), as it performs well in our experiments:

\[
\min_Z - M^{(\text{WYS})} \circ \log h(Z) - (1 - A) \circ \log (1 - h(Z)),
\]

where \( h(Z) = h(\mathbf{L} \| \mathbf{R}) = (1 + \exp(\mathbf{L} \times \mathbf{R}^\top))^{-1} \) i.e. \( Z \) concatenates \( \mathbf{L}, \mathbf{R} \in \mathbb{R}^{n \times \frac{z}{2}} \) and \( h \) is the logistic of their cross-correlation (all pairs dot-product), and

\[
M^{(\text{WYS})} = \sum_i (D^{-1} A)^i c_i = \sum_i T^i c_i,
\]

where \( T \) is the (random walk) transition matrix, \( D = \text{diag}(\mathbf{1}^\top \mathbf{A}) \) is diagonal degree matrix, and we fix vector \( c \) to staircase: \( c_i = \frac{C-1}{C-1+i} \). For instance, \( c = [1, \frac{3}{4}, \frac{2}{4}, \frac{1}{4}] \) for context window size \( C = 4 \).

### 3. Our Proposed Convex Objectives

#### 3.1. Network Embedding

**3.1.1. Model**

Objective in Eq 5 learns node embeddings \( Z = [\mathbf{L} \| \mathbf{R}] \in \mathbb{R}^{n \times \frac{z}{2}} \) using cross-entropy. The terms: model output (outer product, \( \sigma(\mathbf{L} \times \mathbf{R}^\top) \)), negatives (non-edges, \( 1 - A \)), and positives (expected number of node pairs co-occurrences, \( M \)), are all dense matrices with \( n^2 \gg m \) nonzero entries. For instance, even a relatively-small social network with \( n=100,000 \) and average degree of 100 (i.e. \( m=100n \)) would produce an \( M \) occupying \( \approx 40\text{GB} \) memory, whereas one can do the entire learning with \( \approx 40\text{MB} \) memory using functional SVD (§4). We start by designing a matrix \( \hat{M}^{(\text{WYS})} \) incorporating positive and negative information \( (M^{(\text{WYS})} \) and \( 1 - A \)) as:

\[
\hat{M}^{(\text{WYS})} = M^{(\text{WYS})} - \lambda (1 - A) = \sum_i T^i c_i - \lambda (1 - A),
\]

with coefficient \( \lambda \geq 0 \) weighing negative samples. We use \( \hat{\cdot} \) to denote our convexification.

#### 3.1.2. Learning

We can directly set \( Z \) using the SVD basis \( (\mathbf{U}, \mathbf{S}, \mathbf{V}) \). Our matrix \( \hat{M}^{(\text{WYS})} \) has large entry \( \hat{M}_{uv} \) when nodes \( (u, v) \) are well-connected (co-visited many times, during random walks) and small if they are non-edges. SVD provides a rank \( k \) estimator of \( M \) as \( LR^\top \approx M \) i.e. with minimum Frobenius norm of error. As such, we can set the network embedding model parameters \( Z = [\hat{L} \| \hat{R}] \) as:

\[
\text{SVD} (\hat{M}, k) \leftarrow \underset{\hat{U}, \hat{S}, \hat{V}}{\text{arg min}} \left\| \hat{M} - \hat{U} \hat{S} \hat{V}^\top \right\|_F^2
\]

\[
\approx \underset{\hat{U}, \hat{S}, \hat{V}}{\text{arg min}} \left\| \hat{M} - \left( \hat{U} \hat{S} \hat{V}^\top \right) \right\|_F^2
\]

Learning algorithm via SVD trivially follows:

\[
\hat{U}_k, \hat{S}_k, \hat{V}_k \leftarrow \text{SVD} (\hat{M}^{(\text{WYS})}, k)
\]

\[
\hat{L} \leftarrow \hat{U}_k \hat{S}_k^2, \\
\hat{R} \leftarrow \hat{V}_k \hat{S}_k^2,
\]

#### 3.1.3. Inference

Given query edges \( Q = \{(u_i, v_i)\}_i \), model can be computed

\[
H_Q = R^\top_{\{v_i\}}, L_{\{u_i\}},
\]

where the (RHS) set-subscript denotes gathering rows (a.k.a, advanced indexing), and \( H_Q \in \mathbb{R}^{\|Q\|} \).

### 3.2. Messaging Passing Models

#### 3.2.1. Model

We can linearize message passing models (Eq. 3) by assuming all \( \sigma’ \)'s are identity (\( \sigma’(\cdot) = \cdot \)). To simplify the presentation (though not necessary), let \( g = g_1 = g_2 = \ldots \), specifically let \( g(A) = (D + I)^{−1/2}(A + I)(D + I)^{−1/2} \) per renormalization trick of \( (\text{Kipf \\& Welling, } 2017) \). A linear \( L \)-layer message passing network can be:

\[
\hat{H} = \left[ \begin{array}{c} X \\ g(A)X \\ g(A)^2X \\ \ldots \\ g(A)^LX \end{array} \right] \hat{W},
\]

\[
\hat{W} = \begin{bmatrix} \hat{W}_0 & \hat{W}_1 & \ldots \end{bmatrix}
\]

where concatenation of all layers at the output has also appeared in Jumping Knowledge Networks \( (\text{JKN, Xu et al., } 2018) \). In this case, one can reinterpret \( \hat{W} \) as row-wise concatenation of \( L + 1 \) matrices: \( \hat{W}^\top = \left[ \hat{W}_0^\top \ | \ \hat{W}_1^\top \ | \ \ldots \right] \).
3.2.2. LEARNING

We can optimize \( \hat{\mathbf{W}} \) with a convex objective:

\[
\arg \min_{\hat{\mathbf{W}}} \| \mathbf{M}^{(\text{JKN})} \hat{\mathbf{W}} - \mathbf{Y} \|_F^2 \quad (14)
\]

\[
= \arg \min_{\hat{\mathbf{W}}} \| \mathbf{M}^{(\text{JKN})} \hat{\mathbf{W}} - \mathbf{Y} \|_F^2 \quad (15)
\]

Loss in Equation 15 can perform well on classification tasks, and according to Hui & Belkin (2021), as well as the cross entropy loss defined in Equation 4. Taking \( \nabla_{\hat{\mathbf{W}}} \) of Eq. 15 then setting to zero, yields closed-form minimizer for Eq. 15:

\[
\hat{\mathbf{W}}^* \triangleq \arg \min_{\hat{\mathbf{W}}} \| \mathbf{M}^{(\text{JKN})} \hat{\mathbf{W}} - \mathbf{Y} \|_F^2 = \left( \mathbf{M}^{(\text{JKN})} \right) ^\dagger \mathbf{Y}. \quad (16)
\]

Rank-\( k \) SVD can estimate \( \left( \mathbf{M}^{(\text{JKN})} \right)^\dagger \) and hence \( \hat{\mathbf{W}}^* \) as:

\[
\hat{\mathbf{U}}_k, \hat{\mathbf{S}}_k, \hat{\mathbf{V}}_k \leftarrow \text{SVD} (\mathbf{M}^{(\text{JKN})}, k) \quad (17)
\]

\[
\left( \mathbf{M}^{(\text{JKN})} \right)^\dagger \approx \hat{\mathbf{V}}_k \hat{\mathbf{S}}_k^{-1} \hat{\mathbf{U}}_k^\top \quad (18)
\]

\[
\hat{\mathbf{W}}^* \leftarrow (\hat{\mathbf{V}}_k (\hat{\mathbf{S}}_k^{-1} (\hat{\mathbf{U}}_k^\top \mathbf{Y}))). \quad (19)
\]

Order of multiplications in Eq. 19 is for efficiency. Further, in the case when only subset of nodes \( \mathcal{V} = \{ v \}_v \) have labels, the right-most multiplication of Eq. 19 could restricted to the labeled nodes. Let \( \mathbf{Y}_\mathcal{V} \) be a matrix of \( |\mathcal{V}| \) rows selected from \( \mathbf{Y} \) according to elements \( \mathcal{V} \). The right-most multiplication of Eq. 19 can modified to: \( \hat{\mathbf{U}}_{k_\mathcal{V}} Y_{\mathcal{V}} \).

In practice, one can get creative in designing \( \mathbf{M}^{(\text{JKN})} \), for instance, extending \( \mathbf{M} \) with a column of ones \( \mathbf{1} \), to account for offset/bias. For transductive settings, also with powers of transformed adjacency matrix: \( \mathbf{I}, g_1(\mathbf{A}), g_2(\mathbf{A})^2, \ldots \)

3.2.3. NORM REGULARIZATION OF WIDE MODELS

If \( \mathbf{M} \) is too wide, then we need not to worry much about overfitting, due to the following Theorem.

**Theorem 1** (Min. Norm) If system \( \mathbf{M} \hat{\mathbf{W}} = \mathbf{Y} \) is under-determined\(^1\) with rows of \( \mathbf{M} \) being linearly independent, then there are infinitely many solutions. Denote solution space \( \mathcal{W}^* = \{ \hat{\mathbf{W}} \mid \mathbf{M} \hat{\mathbf{W}} = \mathbf{Y} \} \). Then, for \( k \geq \text{rank}(\mathbf{M}) \), matrix \( \hat{\mathbf{W}}^* \), defined in Eq. 19 satisfies:

\[
\hat{\mathbf{W}}^* = \arg \min_{\hat{\mathbf{W}} \in \mathcal{W}^*} \| \hat{\mathbf{W}} \|_F^2. \quad (20)
\]

**Proof** Assume \( \mathbf{Y} = \mathbf{y} \) is a column vector (the proof can be generalized to matrix \( \mathbf{Y} \) by repeated column-wise application\(^2\)). SVD(\( \mathbf{M}, k \)), \( k \geq \text{rank}(\mathbf{M}) \), recovers the solution:

\[
\hat{\mathbf{W}}^* = \left( \mathbf{M} \right)^\dagger \mathbf{y} = \mathbf{M}^\top \left( \mathbf{M} \mathbf{M}^\top \right)^{-1} \mathbf{y}. \quad (21)
\]

The Gram matrix \( \mathbf{M} \mathbf{M}^\top \) is nonsingular as the rows of \( \mathbf{M} \) are linearly independent. To prove the theorem, let us first verify that \( \hat{\mathbf{W}}^* \in \mathcal{W}^* \):

\[
\mathbf{M} (\hat{\mathbf{W}}_p - \hat{\mathbf{W}}^*) = 0. \quad (22)
\]

It follows that \( (\hat{\mathbf{W}}_p - \hat{\mathbf{W}}^*) \perp \hat{\mathbf{W}}^* \):

\[
(\hat{\mathbf{W}}_p - \hat{\mathbf{W}}^*)^\top \hat{\mathbf{W}}^* = (\hat{\mathbf{W}}_p - \hat{\mathbf{W}}^*)^\top \mathbf{M}^\top \left( \mathbf{M} \mathbf{M}^\top \right)^{-1} \mathbf{y} = (\mathbf{M}(\hat{\mathbf{W}}_p - \hat{\mathbf{W}}^*))^\top \left( \mathbf{M} \mathbf{M}^\top \right)^{-1} \mathbf{y} = 0 \quad (23)
\]

Finally, using Pythagoras Theorem (due to \( \perp \)):

\[
\| \hat{\mathbf{W}}_p \|_2^2 = \| \hat{\mathbf{W}}^* + \hat{\mathbf{W}}_p - \hat{\mathbf{W}}^* \|_2^2 = \| \hat{\mathbf{W}}^* \|_2^2 + \| \hat{\mathbf{W}}_p - \hat{\mathbf{W}}^* \|_2^2 \geq \| \hat{\mathbf{W}}^* \|_2^2 \quad \blacksquare
\]

As a consequence, solutions recovered by SVD follow a strong standard Gaussian prior, which may be regarded as a form of regularization.

4. Functional Singular Value Decomposition

4.1. Calculating the SVD

SVD of \( \mathbf{M} \) yields its left and right singular orthonormal (basis) vectors, respectively, in columns of \( \mathbf{U} \) and \( \mathbf{V} \). Since \( \mathbf{U} \) and \( \mathbf{V} \), respectively, are the eigenvectors of \( \mathbf{M} \mathbf{M}^\top \) and \( \mathbf{M}^\top \mathbf{M}\), then perhaps the most intuitive algorithms for SVD are variants of the power iteration, including Arnoldi iteration (Arnoldi, 1951) and Lanczos algorithm (Lanczos, 1950). In practice, randomized algorithms for estimating SVD run faster than these variants, including the algorithm of Halko et al. (2009) which is implemented in scikit-learn. None of these methods require individual access to \( \mathbf{M} \)'s entries, but rather, require two operations: ability to multiply any vector with \( \mathbf{M} \) and with \( \mathbf{M}^\top \). Therefore, it is only a practical gap that we fill in this section: we open-source a TensorFlow implementation that accept product and transpose operators.

\(^1\)E.g., if the number of labeled examples i.e. height of \( \mathbf{M} \) and \( \mathbf{Y} \) is smaller than the width of \( \mathbf{M} \).

\(^2\)The minimizer for the Frobenius norm is composed, column-wise, of the minimizers \( \arg \min_{k \mathbf{W}_{i,j} = \mathbf{y}_{i,j}} \| \mathbf{W}_{i,j} \|_2 \) for all \( j \).
Algorithm 1: Functional Randomized SVD, following prototype and recommendations of Halko et al. (2009)

1: input: rank \( k \in \mathbb{N}_+ \), product fn \( f: \mathbb{R}^m \rightarrow \mathbb{R}^c \), shape fn \( s \), transpose fn \( t: (\mathbb{R}^c \rightarrow \mathbb{R}^r) \rightarrow (\mathbb{R}^r \rightarrow \mathbb{R}^t) \)
2: def fSVD\((f, t, s, k)\):
3: \((r, c) \leftarrow s()\)
4: \(Q \sim \mathcal{N}(0, 1)^{c \times 2k} \) # IID Gaussian. Shape: \((c \times 2k)\)
5: for \( i \leftarrow 1 \) to iters do:
6: \(Q_i \leftarrow tf.linalg.qr(f(Q))\) # \((r \times 2k)\)
7: \(Q_i \leftarrow tf.linalg.qr((t \circ f)(Q))\) # \((c \times 2k)\)
8: \(Q_i \leftarrow tf.linalg.qr(f(Q))\) # \((r \times 2k)\)
9: \(B \leftarrow ((t \circ f)(Q))^\top\) # \((2k \times c)\)
10: \(U, s, V^\top \leftarrow tf.linalg.svd(B)\)
11: \(U \leftarrow Q \times U\) # \((r \times 2k)\)
12: return \(U[:, k], s[:, k], V[:, k]^\top\)

4.2. Implementation

Since we do not explicitly calculate the \( \hat{M} \) matrices displayed in Equations 7 and 13, as doing so consumes quadratic memory \( \mathcal{O}(n^2) \), we implement a functional form SVD of the celebrated randomized SVD algorithm of Halko et al. (2009). To run our Algorithm 1, one must specify \( k \in \mathbb{N}_+ \) (rank of decomposition), as well as functions \( f, l, s \) that the program provider promises they operate as:

1. Product function \( f \) that exactly computes \( f(v) = \langle M, v \rangle \) for any \( v \in \mathbb{R}^c \) (recall: \( M \in \mathbb{R}^{r \times c} \))
2. Transpose\(^1\) function \( t. \forall v \in \mathbb{R}^r, (t \circ f)(v) = \langle M^\top, v \rangle \)
3. Shape (constant) function \( s \) that knows and returns \((r, c)\).

Once transposes, should return \((c, r)\).

To complete our exposition, we need the (straightforward) implementations of the above functions for \( \hat{M}^{(\text{wys})} \) and \( \hat{M}^{(\text{kn})} \). For \( \hat{M}^{(\text{wys})} \): the non-edges term, \((I - A)\), can be trivially re-written by explicit broadcasting \((11^\top - A)\) as:

\[
f^{(\text{wys})}_{\hat{M}}(v) = \sum_i (T_i^\top v c_i - \lambda 1(1^\top v) + \lambda A v. \tag{23}
\]

All matrix-vector products can be efficiently computed when \( A \) is sparse. For instance, computing \( T^\top v \) can be calculated right-to-left as \( T(T^\top T(v)) \) using 3\( m \) floating-point multiplications, consuming no-more than \( \mathcal{O}(m) \) memory. For space constraints, the other (trivial) implementations for \( \hat{M}^{(\text{wys})} \) and \( \hat{M}^{(\text{kn})} \) are left outside this write up\(^4\).

4.3. Analysis: Computational Complexity and Error

Theorem 2 (Linear Time) Functional SVD (Alg. 1) trains our convexified GRL models in time linear in the graph size.

Proof of Theorem 2 for our two model families:
1. For rank-\( k \) SVD over \( f^{(\text{wys})}_{\hat{M}} \): Let cost of running \( f^{(\text{wys})}_{\hat{M}} = T_{\text{mult}} \). The run-time to compute SVD, as derived in Section 1.4.2 of (Halko et al., 2009), is:

\[
\mathcal{O}(kT_{\text{mult}} + (r + c)k^2). \tag{24}
\]

Since \( f^{(\text{wys})}_{\hat{M}} \) can be defined as \( C \) (context window size) multiplications with sparse \( n \times n \) matrix \( T \) with \( m \) non-zero entries, then running fSVD\((f^{(\text{wys})}_{\hat{M}}, k)\) costs:

\[
\mathcal{O}(knC + nk^2) \tag{25}
\]

2. For rank-\( k \) SVD over \( f^{(\text{kn})}_{\hat{M}} \): Suppose feature matrix contains \( d \)-dimensional rows. One can calculate \( \hat{M}^{(\text{kn})} \in \mathbb{R}^{n \times Ld} \) with \( L \) sparse multiplies in \( \mathcal{O}(Lmd) \). Calculating and running SVD (see Section 1.4.1 of Halko et al., 2009) on \( \hat{M}^{(\text{kn})} \) costs total of:

\[
\mathcal{O}(ndL \log(k) + (n + dL)k^2 + Lmd). \tag{26}
\]

Therefore, training time is linear in \( n \) and \( m \).

Contrast with methods of WYS (Abu-El-Haija et al., 2018) and NetMF (Qiu et al., 2018), which require assembling a dense \( n \times n \) matrix requiring \( \mathcal{O}(n^2) \) time to decompose. One wonders: how far are we from the optimal SVD with a linear-time algorithm? The following bounds the error.

Theorem 3 (Exponentially-decaying Approx. Error) Rank-\( k \) randomized SVD algorithm of Halko et al. (2009) gives an approximation error that can be brought down, exponentially-fast, to no more than twice of the approximation error of the optimal (true) SVD.

Proof is in Theorem 1.2 of Halko et al. (2009).

Consequently, compared to NetMF of (Qiu et al., 2018), which incurs unnecessary estimation error, our estimation error can be brought-down exponentially by increasing the \( \text{iters} \) parameter of Alg. 1.

5. Experiments

5.1. Datasets

We apply our functional SVD on popular datasets that can be trained using our simplified (i.e., convexified) models. Specifically either (1) semi-supervised node-classification datasets, where features are present, or (2) link-prediction datasets where features are absent. It is possible to convexify other setups, e.g., link prediction when node features are present, but we leave this as future work. We run experiments on eight graph datasets:
• Two biological networks: Protein-Protein Interactions (PPI) and Drug-Drug Interactions (ogbl-ddi). The first (/ second) is a large graph where every node is a protein (/ drug) and an edge between two nodes indicate that the two proteins (/ drugs) interact. Processed version of PPI was downloaded from (Grover & Leskovec, 2016). ogbl-ddi was downloaded via python framework of Open Graph Benchmark (OGB, Hu et al., 2020).

• Three citation networks that are extremely popular: Cora, Citeseer, Pubmed. Each node is an article and each (directed) edge implies that an article cites another. Additionally, each article is accompanied with a feature vector (containing NLP-extracted features of the article’s abstract), as well as a label (article type).

• Two collaboration datasets: ca-AstroPh and ca-HepTh, where nodes are researchers and an edge between two nodes indicate that the researchers co-published together at least one article, in the areas Astro-Physics and High Energy Physics.

• ego-Facebook: an ego-centered social network.

For citation networks, we processed node features and labels. For all other datasets, we did not process features during training nor inference. For train/validation/test partitions: we used the splits of Yang et al. (2016) for Citeseer, Cora, Pubmed; we used the splits of Abu-El-Haija et al. (2018) for PPI, Facebook, ca-AstroPh and ca-HepTh; we used the splits of OGB (Hu et al., 2020) for ogbl-ddi. All datasets and statistics are summarized in Table 1. In §5.2 and §5.3, unless otherwise noted, we download authors’ source code from github, modify it to record wall-clock run-time, and run on GPU NVIDIA Tesla k80. Thankfully, downloaded code has one script to run each dataset, or hyperparameters are clearly stated in the source paper. However, in §5.4, rather than re-running the methods, we only report numbers displayed on the OGB’s public leaderboard.

5.2. Semi-supervised Node Classification

We consider a transductive setting where a graph is entirely visible (all nodes and edges). Additionally, some of the nodes are labeled. The goal is to recover the labels of unlabeled nodes. All nodes have feature vectors.

Baselines: We download code of GAT (Veličković et al., 2018), MixHop (Abu-El-Haija et al., 2019), GCNII (Chen et al., 2020) and re-ran them, with slight modifications to record training time. However, for baselines Planetoid (Yang et al., 2016) and GCN (Kipf & Welling, 2017), we copied them from the GCN paper (Kipf & Welling, 2017).

In these experiments, to train our method, we run our functional SVD twice per graph. We take the feature matrix \( \mathbf{X} \) bundled with the datasets, and concatenate to it two matrices, \( \mathbf{L} \) and \( \mathbf{R} \), calculated per Equations 10 and 11: the calculation itself invokes our functional SVD (the first time) on \( f_{\mathbf{M}}^{(\text{wys})} \) with rank \( = 32 \). Hyperparameters of \( f_{\mathbf{M}}^{(\text{wys})} \) are \( \lambda \) (negative coefficient) and \( C \) (context window-size). After concatenating \( \mathbf{L} \) and \( \mathbf{R} \) into \( \mathbf{X} \), we PCA the resulting matrix to 1000 dimensions, which forms our new \( \mathbf{X} \). Finally, we express our model as the linear \( L \)-layer messaging passing network (Eq. 13) and learn its parameters via rank \( k \) SVD on \( f_{\mathbf{M}}^{(\text{bns})} \) (the second time), as explained in §3.2.2. We use the validation partition to tune \( L, k, \lambda, \) and \( C \).

Table 2 summarizes the performance of our approach \( (f_{\mathbf{M}}^{(\text{bns})}) \) against aforementioned baselines, showing both test accuracy and training time. While our method is competitive with state-of-the-art, it trains much faster.

5.3. ROC-AUC Link Prediction

Given a partial graph: only a subset of edges are observed. The goal is to recover unobserved edges. This has applications in recommender systems: when a user expresses interest in products, the system wants to predict other products the user is interested in. The task is usually setup by partitioning the edges of the input graph into train and test edges. Further, it is common to sample negative test edges e.g. uniformly from the graph compliment. Lastly, a GRL method for link prediction can be trained on the train edges partition, then can be asked to score the test partition edges versus the negative test edges. The quality of the scoring can be quantified by a ranking metric, e.g., ROC-AUC.

Baselines: We download code of WYS (Abu-El-Haija et al., 2018) and update it to for TensorFlow-2.0. We download code of Qiu et al. (2018) and denote their methods as NetMF and NetMF, where the first computes complete matrix \( \mathbf{M} \) before SVD decomposition and the second sample \( \mathbf{M} \) entry-wise – the second is faster for larger graphs but sacrifices on estimation error and performance. For node2vec (n2v), we use its PyG implementation (Fey & Lenssen, 2019).

Table 3 summarizes results, including both accuracy and time. For our method (denoted \( f_{\mathbf{M}}^{(\text{wys})} \)), we call our functional SVD (Alg. 1) and pass it \( f_{\mathbf{M}}^{(\text{wys})} \), as defined in Eq. 23. Embeddings are set to the SVD basis (as in, §3.1.2) and edge score of nodes \( (u, v) \) is \( \infty \) dot-product of embeddings (as in, §3.1.3). The last row of Table 3 shows results when svd rank =100. Lastly, we set the context window hyperparameter (a.k.a, length of walk) as follows. For WYS, we trained with their default context (as WYS learns the context), but for all others (NetMF, n2v, ours) we used context window of length \( C=5 \) for datasets Facebook and PPI (for us, this sets \( c = [1, \frac{4}{5}, \frac{3}{5}, \frac{2}{5}, \frac{1}{5}] \)) and \( C=20 \) for AstroPh and HepTh.

\(^3\)Modified files are in our code repo
Table 1. Dataset Statistics

| Dataset   | Nodes   | Edges                | Source                           |
|-----------|---------|----------------------|----------------------------------|
| PPI       | 3,852 proteins | 20,881 chemical interactions | [http://snap.stanford.edu/node2vec/](http://snap.stanford.edu/node2vec/) |
| ego-Facebook | 4,039 users       | 88,234 friendships    | [http://snap.stanford.edu/data](http://snap.stanford.edu/data) |
| ca-AstroPh | 17,903 researchers | 197,031 co-authorships | [http://snap.stanford.edu/data](http://snap.stanford.edu/data) |
| ca-HepTh  | 8,638 researchers | 24,827 co-authorships  | [http://snap.stanford.edu/data](http://snap.stanford.edu/data) |
| Cora      | 2,708 articles  | 5,429 citations       | Planetoid (Yang et al., 2016)    |
| Citeseer  | 3,327 articles  | 4,732 citations       | Planetoid (Yang et al., 2016)    |
| Pubmed    | 19,717 articles | 44,338 citations      | Planetoid (Yang et al., 2016)    |
| ogbl-ddi  | 4,267 drugs     | 1,334,889 interactions | Open Graph Benchmark (Hu et al., 2020) |

Figure 1. ROC-AUC versus train time of methods on datasets. Each dataset has a distinct shape, with shape size proportional to graph size. Each method uses a different color. Our methods are in blue (dark uses SVD rank $k = 32$, light uses $k = 100$, trading estimation accuracy for train time). Ideal methods should be placed on top-left corner (i.e., higher test ROC-AUC and faster training time).

Table 2. Test accuracy (training wall-clock time) for Semi-supervised Node Classification, over citation datasets.

| Dataset  | Cora  | Citeseer | Pubmed   |
|----------|-------|----------|----------|
| Planetoid| 75.7 (13s) | 64.7 (26s) | 77.2 (25s) |
| GCN      | 81.5 (4s)  | 70.3 (7s)  | 79.0 (83s)  |
| GAT      | 83.2 (1m23s) | 72.4 (3m27) | 77.7 (5m33s) |
| MixHop   | 81.9 (26s)  | 71.4 (31s)  | 80.8 (1m16s) |
| GCNII    | 85.5 (2m29s) | 73.4 (2m55s) | 80.3 (1m42s) |
| $f_{\hat{M}^{(RBN)}}$ | 82.4 (0.28s) | 72.2 (0.13s) | 79.7 (0.14s) |

Table 2. Test accuracy (training wall-clock time) for link prediction when embedding with $z = 64 = 2k$ dimensions per node.

| Dataset   | Facebook | AstroPh | HepTh | PPI   |
|-----------|----------|---------|-------|-------|
| WYS       | 99.4 (54s) | 97.9 (59m) | 93.6 (4m) | 89.8 (54s) |
| n2v       | 99.0 (30s) | 97.8 (2m)  | 92.3 (55s)  | 83.1 (27s)  |
| NetMF     | 97.6 (5s)  | 96.8 (9m)  | 90.5 (4m25s) | 73.6 (7s)  |
| $\tilde{M}_{\text{NetMF}}$ | 97.0 (4s)  | 81.9 (4m24s) | 85.0 (48s)  | 63.6 (10s)  |
| $f_{\hat{M}^{(WYS)}}$ (k=100) | 98.7 (0.32s) | 92.1 (1.9s) | 89.2 (0.64s) | 87.9 (0.3s) |

Figure 2. Test Accuracy VS Depth of model defined by $\hat{M}^{(RBN)}$.

Figure 3. Test ROC-AUC VS rank of SVD on $\hat{M}^{(WYS)}$.
5.4. Hits@20 Drug-Drug Interactions

The last dataset we experiment on, that meets the specifications set in beginning of §5.1 is the Drug-Drug Interactions network, made available by the OGB Team (Hu et al., 2020). The dataset comes with node features that our method ignores. We utilize the Evaluator code by OGB to obtain Hits@20 metrics. We do not run any baselines, but rather copy results from the public leaderboard URL\(^6\).

| Model               | Hits@20 (Avg ± Std) |
|---------------------|---------------------|
| MAD (Luo et al., 2021) | 0.678 ± 2.9e−2      |
| LRGA+GCN (Puny et al., 2020) | 0.623 ± 9.1e−2      |
| GCN+JKNet (Xu et al., 2018)  | 0.606 ± 8.7e−2      |
| GraphSAGE (Hamilton et al., 2017) | 0.539 ± 4.7e−2      |
| GCN (Kipf & Welling, 2017)  | 0.371 ± 5e−2        |
| SEAL (Zhang & Chen, 2018)   | 0.306 ± 3.8e−2      |
| DeepWalk (Perozzi et al., 2014) | 0.264 ± 6.1e−2      |
| iSVD(f_{M|WYS}, 80)  | 0.686 ± 2.5e−5      |

\(^6\) https://ogb.stanford.edu/docs/leader_linkprop/#ogbl-ddi

5.5. Sensitivity Analysis

While in §5.2 we tune the number of layers (\(L\)) using the performance on the validation partition, in this section, we show impact of varying \(L\) on test accuracy. According to the summary in Figure 2, accuracy of classifying a node improves when incorporating information from further nodes. We see little gains beyond \(L > 6\). Note that \(L = 0\) corresponds to ignoring the adjacency matrix altogether when running \(f_{M|WYS}\). Here, we fixed \(\lambda = C = 1\) and averaged 5 runs. The (tiny) error bars show the standard deviation.

Further, while in §5.3 we do SVD on \(f_{M|WYS}\) with rank \(k = 32\) or \(k = 100\), Figure 3 shows test accuracy while sweeping \(k \leq 32\). In general, increasing the rank improves estimation accuracy and test performance. However, if \(k\) is larger than the inherit dimensionality of the data, then this could cause overfitting (though perfect memorization of training edges). The Norm Regularization note (§3.2.3) applies only to pseudoinversion i.e. our classification models.

6. Related Work

Our (convexified) message passing (MP) models train fast. Others have also proposed faster training of MP models by either incorporating neighborhood sampling including GraphSAGE (Hamilton et al., 2017), FastGCN (Chen et al., 2018), Adaptive Sampling (Huang et al., 2018), and GTTF (Markowitz et al., 2021). We differ from these methods: rather than sampling, our method is equivalent to the full-batch objective i.e. operating on the entire graph at once. In that regard, this is similar to SimpleGCN (Wu et al., 2019) and SIGN (Frasca et al., 2020) that linearize an MP model (by setting the element-wise activations to identity), on top thereof train a 1- or multi-layer network using cross entropy objectives, i.e. with no known closed-form solutions. Even though we also linearize the MP model, more importantly, we additionally swap the cross-entropy objective with Frobenius norm, and therefore admitting closed-form solutions via functional SVD on \(f_{M|WYS}\).

Our (convexified) network embedding (NE) models trains fast. Specifically, the embeddings \(Z\) are learned by direct assignment to the output of SVD on \(f_{M|WYS}\). Similarly, Chen et al. (2017); Qiu et al. (2018); Abu-El-Haija et al. (2018) also pose NE optimization as decomposition of some matrix \(M\). However, Abu-El-Haija et al. (2018) use cross-entropy objective, and Chen et al. (2017); Qiu et al. (2018) use Frobenius norm objective but rely on vanilla SVD. All these methods require either explicitly calculating \(M\) (dense for real-world networks, per Small World Phenomenon, §2.2), or sampling some entries of \(M\) (while sacrificing error bounds of the obtained solution). However, we rather implement and utilize a functional of the randomized SVD Algorithm of Halko et al. (2009), allowing us to quickly train \(Z\) as-if we had explicitly calculated \(M\), yet without computing it nor sacrificing the accuracy of the decomposition.

7. Conclusion & Future work

We believe that math frameworks (e.g., TensorFlow) have enabled innovative advancements on the field of GRL. We also believe that a functional SVD implementation might (re-)enable simple yet effective methods on a range of GRL tasks where the implicit design matrix is dense but multiplications against it can be efficiently computed with a series of sparse multiplications.

We implement a handful of code lines, invoking our functional, yielding simple yet effective models for network embedding and message passing, setting competitive test performance yet training much faster.

Although convexified models (trained by fSVD) may not outperform deeper networks, we advocate that practitioners should utilize them at least as baselines, initially before spending marvelous efforts in carefully-tailoring model architectures and/or training hyperparameters (such as learning rate, dropout, regularization coefficient, etc). Further, one could use fSVD output to initialize non-linear models.

Future work could incorporate more model families, e.g., link-prediction when node features are present or in temporal graphs (Zhu et al., 2017). Further, fSVD could be made trainable to tune hyperparameters in the implicit matrices, so long the SVD basis was utilized in an upstream objective.
Acknowledgements

This material is based upon work supported by the Defense Advanced Research Projects Agency (DARPA) and the Army Contracting Command-Aberdeen Proving Grounds (ACC-APG) under Contract Number W911NF-18-C-0020.

References

Abadi, M., Agarwal, A., et al. TensorFlow: Large-scale machine learning on heterogeneous systems. In *USENIX Symposium on Operating Systems Design and Implementation*, OSDI, pp. 265–283, 2016.

Abu-El-Haija, S., Perozzi, B., Al-Rfou, R., and Alemi, A. A. Watch your step: Learning node embeddings via graph attention. In *Advances in Neural Information Processing Systems*, NeurIPS, 2018.

Abu-El-Haija, S., Perozzi, B., Kapoor, A., Harutyunyan, H., Alipourfard, N., Lerman, K., Steeg, G. V., and Galstyan, A. Mixhop: Higher-order graph convolutional architectures via sparsified neighborhood mixing. In *International Conference on Machine Learning*, ICML, 2019.

Arnoldi, W. E. The principle of minimized iterations in the solution of the matrix eigenvalue problem. In *Quarterly of Applied Mathematics*, pp. 17–29, 1951.

Backstrom, L., Boldi, P., Rosa, M., Ugander, J., and Vigna, S. Four degrees of separation. In *ACM Web Science Conference*, pp. 33–42, 2012.

Belkin, M. and Niyogi, P. Laplacian eigenmaps for dimensionality reduction and data representation. In *Neural Computation*, pp. 1373–1396, 2003.

Chen, J., Ma, T., and Xiao, C. FastGCN: Fast learning with graph convolutional networks via importance sampling. In *International Conference on Learning Representations*, 2018.

Chen, M., Wei, Z., Huang, Z., Ding, B., and Li, Y. Simple and deep graph convolutional networks. In *International Conference on Machine Learning*, ICML, 2020.

Chen, S., Niu, S., Akoglu, L., Kovacevic, J., and Faloutsos, C. Fast, warped graph embedding: Unifying framework and one-click algorithm. *arxiv:1702.05764*, 2017.

Deerwester, S., Dumais, S. T., Furnas, G. W., Landauer, T. K., and Harshman, R. Indexing by latent semantic analysis. In *Journal of the American Society for Information Science*, pp. 391–407, 1990.

Eckart, C. and Young, G. The approximation of one matrix by another of lower rank. In *Psychometrika*, pp. 211–218, 1936.

Fey, M. and Lenssen, J. E. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

Frasca, F., Rossi, E., Eynard, D., Chamberlain, B., Bronstein, M., and Monti, F. SIGN: Scalable inception graph neural networks. In *Graph Representation Learning and Beyond (GRL+) Workshop at the International Conference on Machine Learning*, GRL+ @ ICML, 2020.

Golub, G. H. and Loan, C. F. V. *Matrix Computations*, pp. 257–258. John Hopkins University Press, Baltimore, 3rd edition, 1996.

Grover, A. and Leskovec, J. Node2vec: Scalable feature learning for networks. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD, pp. 855–864, 2016.

Halko, N., G., M. P., and Tropp, J. A. Finding structure with randomness: Stochastic algorithms for constructing approximate matrix decompositions. In *ACM Technical Reports*, 2009.

Hamilton, W., Ying, R., and Leskovec, J. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, NeurIPS, 2017.

Hu, W., Fey, M., Zitnik, M., Dong, Y., Ren, H., Liu, B., Catasta, M., and Leskovec, J. Open graph benchmark: Datasets for machine learning on graphs. In *arXiv*, 2020.

Huang, W., Zhang, T., Rong, Y., and Huang, J. Adaptive sampling towards fast graph representation learning. In *Advances in Neural Information Processing Systems*, NeurIPS, pp. 4558–4567, 2018.

Hui, L. and Belkin, M. Evaluation of neural architectures trained with square loss vs cross-entropy in classification tasks. In *International Conference on Learning Representations*, 2021.

Karinthy, F. Chain links. In *Everything is Different*, 1929.

Kipf, T. and Welling, M. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, ICLR, 2017.

Koren, Y., Bell, R., and Volinsky, C. Matrix factorization techniques for recommender systems. In *IEEE Computer*, pp. 30–37, 2009.

Lanczos, C. An iteration method for the solution of the eigenvalue problem of linear differential and integral operators. In *Journal of Research of the National Bureau of Standards*, pp. 255–282, 1950.
Levy, O. and Goldberg, Y. Neural word embedding as implicit matrix factorization. In *Advances in Neural Information Processing Systems*, NeurIPS, pp. 2177–2185, 2014.

Luo, Y., Chen, A., Hui, B., and Yan, K. Memory-associated differential learning, 2021.

Markowitz, E. S., Balasubramanian, K., Mirtaheri, M., Abu-El-Haija, S., Perozzi, B., Steeg, G. V., and Galstyan, A. Graph traversal with tensor functionals: A meta-algorithm for scalable learning. In *International Conference on Learning Representations*, ICLR, 2021.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*, NeurIPS, pp. 3111–3119, 2013.

Paszke, A., Gross, S., Massa, F., and Others. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, 2019.

Pennington, J., Socher, R., and Manning, C. GloVe: Global vectors for word representation. In *Empirical Methods in Natural Language Processing*, EMNLP, pp. 1532–1543, 2014.

Perozzi, B., Al-Rfou, R., and Skiena, S. Deepwalk: Online learning of social representations. In *ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD, pp. 701–710, 2014.

Puny, O., Ben-Hamu, H., and Lipman, Y. Global attention improves graph networks generalization, 2020.

Qiu, J., Dong, Y., Ma, H., Li, J., Wang, K., and Tang, J. Network embedding as matrix factorization: Unifying deepwalk, line, pte, and node2vec. In *International Conference on Web Search and Data Mining*, WSDM, pp. 459–467, 2018.

Travers, J. and Milgram, S. An experimental study of the small world problem. In *Sociometry*, pp. 425–443, 1969.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., and Bengio, Y. Graph attention networks. In *International Conference on Learning Representations*, ICLR, 2018.

Wu, F., Souza, A., Zhang, T., Fifty, C., Yu, T., and Weinberger, K. Simplifying graph convolutional networks. In *International Conference on Machine Learning*, ICML, 2019.

Xu, K., Li, C., Tian, Y., Sonobe, T., Kawarabayashi, K.-i., and Jegelka, S. Representation learning on graphs with jumping knowledge networks. In *International Conference on Machine Learning*, ICML, pp. 5453–5462, 2018.

Xu, K., Hu, W., Leskovec, J., and Jegelka, S. How powerful are graph neural networks? In *International Conference on Learning Representations*, ICLR, 2019.

Yang, Z., Cohen, W. W., and Salakhutdinov, R. Revisiting semi-supervised learning with graph embeddings. In *International Conference on Machine Learning*, 2016.

Zhang, M. and Chen, Y. Link prediction based on graph neural networks. In *Advances in Neural Information Processing Systems*, pp. 5165–5175, 2018.

Zhu, L., Guo, D., Yin, J., Ver Steeg, G., and Galstyan, A. Scalable temporal latent space inference for link prediction in dynamic social networks (extended abstract). In *IEEE International Conference on Data Engineering*, ICDE, 2017.