FAST GRL WITH UNIQUE OPTIMAL SOLUTIONS

Anonymous authors
Paper under double-blind review

ABSTRACT

We consider two popular Graph Representation Learning (GRL) methods: message passing for node classification and network embedding for link prediction. For each, we pick a popular model that we: (i) *linearize* and (ii) and switch its training objective to *Frobenius norm error minimization*. These simplifications can cast the training into finding the optimal parameters in closed-form. We program in TensorFlow a functional form of Truncated Singular Value Decomposition (SVD), such that, we could decompose a dense matrix $\mathbf{M}$, without explicitly computing $\mathbf{M}$. We achieve competitive performance on popular GRL tasks while providing orders of magnitude speedup. We open-source our code at `http://anonymous/url/for/review`

1 INTRODUCTION

Many recent graph representation learning (GRL) models are creative and theoretically-justified (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, however, they 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.

We circumvent these weaknesses. We (i) **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, Singlar Value Decomposition (SVD).

SVD periodically appears within powerful yet simple methods, competing on state-of-the-art. The common practice is to design a matrix $\mathbf{M}$, such that its decomposition (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 $\mathbf{M}$. However, their decomposition requires $\mathbf{M}$ 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, (Travers & Milgram, 1969)) and (b) incurs unnecessary estimation errors. On the other hand, known algorithms in matrix theory can decompose *any* matrix $\mathbf{M}$ *without explicitly knowing* $\mathbf{M}$. Specifically, it is sufficient to provide a function $f_{\mathbf{M}}(.) = \langle \mathbf{M}, . \rangle$ that can multiply $\mathbf{M}$ with arbitrary vectors (§4). We, argue that if the popular frameworks (e.g., TensorFlow) implement a *functional SVD*, that accept $f_{\mathbf{M}}(.)$ rather than $\mathbf{M}$, then modern practitioners may find it useful.

We review powerful GRL methods (§2.2) that we *convexify* (§3), allowing us to use (randomized) SVD for obtaining (approximate) optimum solutions. Our contributions are:

1. We implement a functional SVD (§4) of the randomized algorithm of Halko et al. (2009).
2. We approximate embedding and message passing methods via SVD (§3), showing competitive performance with state-of-the-art, yet much faster to train (§5).
3. We analyze that learning is fast and approximation error can be made arbitrarily small (§A.3.2).

2 PRELIMINARIES

2.1 SINGULAR VALUE DECOMPOSITION (SVD)

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

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

while parameterizing $\widetilde{\mathbf{M}}$ as $\widetilde{\mathbf{M}} = \mathbf{U}_k \mathbf{S}_k \mathbf{V}_k^\top$, subject to, columns of $\mathbf{U}_k, \mathbf{V}_k$ being orthonormal. It turns out, the minimizer of Frobenius norm $||.||_F$ recovers the top-k singular values (stored along diagonal matrix $\mathbf{S}_k \in \mathbb{R}^{k \times k}$), with their corresponding left- and right-singular vectors, respectively, stored as columns of the unitary matrices $\mathbf{U}_k \in \mathbb{R}^{r \times k}$ and $\mathbf{V}_k \in \mathbb{R}^{c \times k}$ (*a.k.a*, the *singular bases*).

SVD has many applications and we utilize two: (1) it is used for embedding and matrix completion; and (2) it can estimate the pseudoinverse of matrix $\mathbf{M}$ (a.k.a., Moore-Penrose inverse), as:

$$\mathbf{M}^\dagger \triangleq \mathbf{M}^\top \left(\mathbf{M}\mathbf{M}^\top\right)^{-1} \approx \mathbf{V}_k \mathbf{S}_k^{-1} \mathbf{U}_k^\top, \tag{2}$$

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

2.2 GRAPH REPRESENTATION LEARNING (GRL)

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

$$\mathbf{H} = \sigma_L\Bigg(g_L(\mathbf{A}) \dots \overbrace{\sigma_2\Bigg(g_2(\mathbf{A}) \underbrace{\sigma_1\left(g_1(\mathbf{A})\mathbf{X}\mathbf{W}_1\right)}_{\text{output of layer 1}} \mathbf{W}_2\Bigg)}^{\text{output of layer 2}} \dots \mathbf{W}_L\Bigg) \tag{3}$$

where L is the number of layers, matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$ contains d features per node, $\mathbf{W}$'s are trainable parameters, σ 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 (Veličković et al., 2018) set $g(\mathbf{A}) = \mathbf{A} \circ \text{MultiHeadedAttention}$ and GIN (Xu et al., 2019) as $g(\mathbf{A}) = \mathbf{A} + (1+\epsilon)\mathbf{I}$ with identity $\mathbf{I}$ and $\epsilon > 0$. For node classification, it is common to set $\sigma_L = \text{softmax}$ (applied row-wise), specify the size of $\mathbf{W}_L$ s.t. $\mathbf{H} \in \mathbb{R}^{n \times y}$ where y is number of classes, and optimize cross-entropy objective:

$$\min_{\{\mathbf{W}_j\}_{j=1}^L} -\mathbf{Y} \circ \log \mathbf{H} - (1-\mathbf{Y}) \circ \log(1-\mathbf{H}), \tag{4}$$

where $\mathbf{Y}$ 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 $\mathbf{Y}$ and $\mathbf{H}$ that correspond to labeled nodes.

(ii) *Network embedding* methods map nodes onto a z-dimensional vector space $\mathbf{Z} \in \mathbb{R}^{n \times z}$. Modern approaches train skipgram models (e.g. word2vec (Mikolov et al., 2013)) on sampled random walks. It has been shown that these skipgram network embedding methods, including DeepWalk (Perozzi et al., 2014)) and node2vec (Grover & Leskovec, 2016), with a learning process of walk sampling followed by positional embedding, can be approximated as a matrix deomposition (Chen et al., 2017; Abu-El-Haija et al., 2018; Qiu et al., 2018). We point the curious reader to the listed papers for how the decomposition was derived, but show here the derivation of Abu-El-Haija et al. (WYS, 2018), as it performs well in our experiments:

$$\min_{\mathbf{Z}} -\mathbf{M}^{(\text{WYS})} \circ \log h(\mathbf{Z}) - (1-\mathbf{A}) \circ \log(1-h(\mathbf{Z})), \tag{5}$$

where $h(\mathbf{Z}) = h([\mathbf{L} \;\vdots\; \mathbf{R}]) = (1+\exp(\mathbf{L} \times \mathbf{R}^\top))^{-1}$ i.e. $\mathbf{Z}$ concatenates $\mathbf{L}, \mathbf{R} \in \mathbb{R}^{n \times \frac{z}{2}}$ and h is the logistic of their cross-correlation (pairwise dot-products). $\mathbf{M}^{(\text{WYS})} = \sum_i^C (\mathbf{D}^{-1}\mathbf{A})^i \mathbf{c}_i = \sum_{i=1}^C \mathcal{T}^i \mathbf{c}_i$, where $\mathcal{T}$ is the transition matrix, $\mathbf{D} = \text{diag}(\mathbf{1}^\top \mathbf{A})$ is diagonal degree matrix, and we fix vector $\mathbf{c}$ to staircase: $\mathbf{c}_i = \frac{C-i+1}{C}$. For instance, $\mathbf{c} = [1, \frac{3}{4}, \frac{2}{4}, \frac{1}{4}]$ for context size $C = 4$.

3 OUR PROPOSED CONVEX OBJECTIVES

3.1 NETWORK EMBEDDING MODEL

Objective in Eq 5 learns node embeddings $\mathbf{Z} = [\mathbf{L} \mid \mathbf{R}] \in \mathbb{R}^{n \times z}$ using cross-entropy. The terms: model output (outer product, $\sigma(\mathbf{L} \times \mathbf{R}^\top)$), negatives (non-edges, $1 - \mathbf{A}$), and positives (expected number of node pairs covisits, $\mathbf{M}$), 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 $\mathbf{M}$ occupying ≈40GB memory, whereas one can do the entire learning with ≈40MB memory using functional SVD (§4). We start by designing a matrix $\widehat{\mathbf{M}}^{(\text{WYS})}$ incorporating positive and negative information ($\mathbf{M}^{(\text{WYS})}$ and $1 - \mathbf{A}$) as:

$$\widehat{\mathbf{M}}^{(\text{WYS})} = \mathbf{M}^{(\text{WYS})} - \lambda(1 - \mathbf{A}) = \sum_i \mathcal{T}^i \mathbf{c}_i - \lambda(1 - \mathbf{A}), \tag{6}$$

with coefficient $\lambda \geq 0$ weighing negative samples. We use $\widehat{\cdot}$ to denote our convexification.

Learning: We can directly set $\mathbf{Z}$ to the SVD *basis* $(\mathbf{U}, \mathbf{S}, \mathbf{V})$. Matrix $\widehat{\mathbf{M}}^{(\text{WYS})}$ has large entry $\widehat{\mathbf{M}}_{uv}$ 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 $\widehat{\mathbf{M}}$ as $\widehat{\mathbf{L}}\widehat{\mathbf{R}}^\top \approx \widehat{\mathbf{M}}$ i.e. with minimum Frobenius norm of error. We can set the network embedding model parameters $\mathbf{Z} = [\widehat{\mathbf{L}} \mid \widehat{\mathbf{R}}]$ as:

$$\text{SVD}(\widehat{\mathbf{M}}, k) \leftarrow \underset{\widehat{\mathbf{U}}_k \widehat{\mathbf{S}}_k \widehat{\mathbf{V}}_k}{\arg\min} \left|\left| \widehat{\mathbf{M}} - \widehat{\mathbf{U}}_k \widehat{\mathbf{S}}_k \widehat{\mathbf{V}}_k^\top \right|\right|_F^2 = \underset{\widehat{\mathbf{U}}_k \widehat{\mathbf{S}}_k \widehat{\mathbf{V}}_k}{\arg\min} \left|\left| \widehat{\mathbf{M}} - \underbrace{\left(\widehat{\mathbf{U}}_k \widehat{\mathbf{S}}_k^{\frac{1}{2}}\right)}_{\triangleq \widehat{\mathbf{L}}} \underbrace{\left(\widehat{\mathbf{S}}_k^{\frac{1}{2}} \widehat{\mathbf{V}}_k^\top\right)}_{\triangleq \widehat{\mathbf{R}}^\top} \right|\right|_F^2 \tag{7}$$

$$\text{Learning follows as:} \quad \widehat{\mathbf{U}}_k, \widehat{\mathbf{S}}_k, \widehat{\mathbf{V}}_k \leftarrow \text{SVD}(\widehat{\mathbf{M}}^{(\text{WYS})}, k); \quad \widehat{\mathbf{L}} \leftarrow \widehat{\mathbf{U}}_k \widehat{\mathbf{S}}_k^{\frac{1}{2}}; \quad \widehat{\mathbf{R}} \leftarrow \widehat{\mathbf{V}}_k \widehat{\mathbf{S}}_k^{\frac{1}{2}} \tag{8}$$

Inference: Given query edges $Q = \{(u_i, v_i)\}_i$, one can compute model: $\mathbf{H}_Q = \widehat{\mathbf{R}}_{\{v\}_i}^\top \widehat{\mathbf{L}}_{\{u\}_i}$, (9)

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

3.2 MESSAGING PASSING MODELS

We can linearize **message passing** models (Eq. 3) by assuming all σ's are identity ($\sigma(.) = .$). Let $g = g_1 = g_2 = \ldots$, specifically let $g(\mathbf{A}) = (\mathbf{D} + \mathbf{I})^{-1/2}(\mathbf{A} + \mathbf{I})(\mathbf{D} + \mathbf{I})^{-1/2}$ per renormalization trick of (Kipf & Welling, 2017). A linear L-layer message passing network can be:

$$\widehat{\mathbf{H}} = \underbrace{\left[\mathbf{X} \mid g(\mathbf{A})\mathbf{X} \mid g(\mathbf{A})^2\mathbf{X} \mid \ldots \mid g(\mathbf{A})^L\mathbf{X}\right]}_{\triangleq\ \widehat{\mathbf{M}}^{(\text{JKN})}} \widehat{\mathbf{W}}. \tag{10}$$

Concatenation of all layers was proposed in Jumping Knowledge Networks (JKN, Xu et al., 2018).

Learning: We optimize $\widehat{\mathbf{W}}$ with: $$\min_{\widehat{\mathbf{W}}} ||\widehat{\mathbf{H}} - \mathbf{Y}||_F^2 = \min_{\widehat{\mathbf{W}}} ||\widehat{\mathbf{M}}^{(\text{JKN})}\widehat{\mathbf{W}} - \mathbf{Y}||_F^2 \tag{11}$$

Loss in Equation 11 can perform well on classification tasks, and according to Hui & Belkin (2021), as well as the cross entropy loss defined in Equation 4. Taking $\nabla_{\widehat{\mathbf{W}}}$ of Eq. 11 then setting to zero,

yields minimizer: $$\widehat{\mathbf{W}}^* \triangleq \underset{\widehat{\mathbf{W}}}{\arg\min} ||\widehat{\mathbf{M}}^{(\text{JKN})}\widehat{\mathbf{W}} - \mathbf{Y}||_F^2 = \left(\widehat{\mathbf{M}}^{(\text{JKN})}\right)^\dagger \mathbf{Y}. \tag{12}$$

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

$$\widehat{\mathbf{U}}_k, \widehat{\mathbf{S}}_k, \widehat{\mathbf{V}}_k \leftarrow \text{SVD}(\widehat{\mathbf{M}}^{(\text{JKN})}, k); \qquad \widehat{\mathbf{W}}^* \leftarrow (\widehat{\mathbf{V}}_k(\widehat{\mathbf{S}}_k^{-1}(\widehat{\mathbf{U}}_k^\top \mathbf{Y}))). \tag{13}$$

Order of multiplications in Eq. 13 is for efficiency. Further, in the case when only subset of nodes $\mathcal{V} = \{v\}_v$ have labels, the right-most multiplication of Eq. 13 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. 13 can modified to: $\widehat{\mathbf{U}}_{k\mathcal{V}}^\top \mathbf{Y}_\mathcal{V}$.

4 Functional Singular Value Decomposition

We do not have to explicitly calculate the matrices $\widehat{\mathbf{M}}$. Rather, we only need to implement product functions $f_{\widehat{\mathbf{M}}}(\mathbf{v}) = \langle \widehat{\mathbf{M}}, \mathbf{v} \rangle$ that can multiply $\widehat{\mathbf{M}}$ with arbitrary (appropriately-sized) vector $\mathbf{v}$. We implement a (TensorFlow) **functional** version of the randomized SVD algorithm of Halko et al. (2009), that accepts $f_{\widehat{\mathbf{M}}}$ rather than $\widehat{\mathbf{M}}$. We show that it can train our models quickly and with arbitrarily small approximation error (in linear time of graph size, in practice, with less than 10 passes over the data) and can yield l2-regularized solutions for classification (see Appendix). We now need the (straightforward $f_{\widehat{\mathbf{M}}^{(\text{WYS})}}$ and $f_{\widehat{\mathbf{M}}^{(\text{JKN})}}$. We leave the second outside this writing. For the first, the non-edges term, $(1 - \mathbf{A})$, can be re-written by explicit broadcasting as $(\mathbf{1}\mathbf{1}^\top - \mathbf{A})$ giving

$$f_{\widehat{\mathbf{M}}^{(\text{WYS})}}(\mathbf{v}) = \sum_i \underbrace{(\mathcal{T})^i \mathbf{v}}_{\mathcal{O}(m)} \mathbf{c}_i - \lambda \mathbf{1} \underbrace{(\mathbf{1}^\top \mathbf{v})}_{\mathcal{O}(n)} + \lambda \mathbf{A}\mathbf{v}. \tag{14}$$

All matrix-vector products can be efficiently computed when $\mathbf{A}$ is sparse.

5 Experimental Results (details in Appendix)

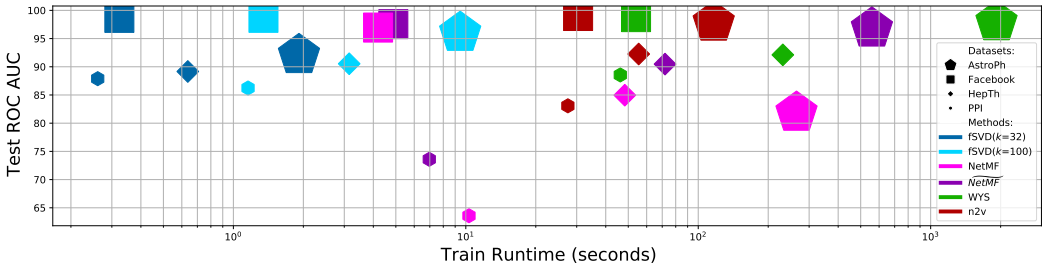

Figure 1: ROC-AUC versus train time of methods on datasets. Each dataset has a distinct shape, with shape size proportional to graph size. Each method uses a different color. Our methods are in blue (dark uses SVD rank $k = 32$, light uses $k = 100$, trading estimation accuracy for train time). Ideal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

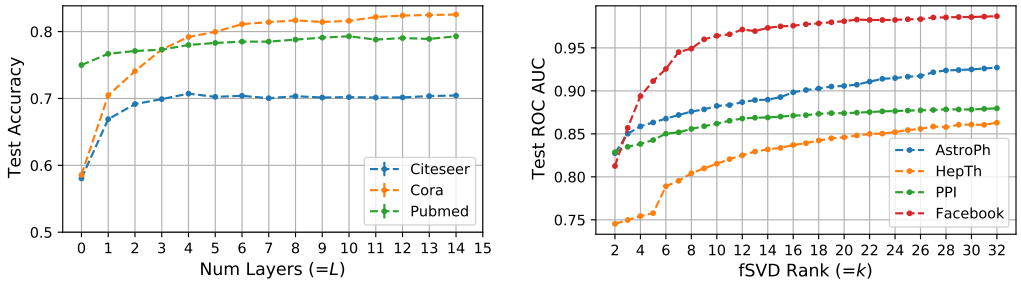

Figure 2: Sensitivity Analysis. **Left**: Test Accuracy VS Depth of model defined by $\widehat{\mathbf{M}}^{(\text{JKN})}$. **Right**: Test ROC-AUC VS rank of SVD on $\widehat{\mathbf{M}}^{(\text{WYS})}$.

Table 1: Test Performance. **Left**: accuracy (training time) for Semi-supervised Node Classification, over citation datasets. **Right**: ROC-AUC for link prediction when embedding with z=64=2k.

	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)
$f_{\widehat{\mathbf{M}}^{(\text{JKN})}}$	82.4 (0.28s)	72.2 (0.13s)	79.7 (0.14s)

	FB	**AstroPh**	**HepTh**	**PPI**
WYS	99.4	97.9	93.6	89.8
n2v	99.0	97.8	92.3	83.1
NetMF	97.6	96.8	90.5	73.6
$\widetilde{\text{NetMF}}$	97.0	81.9	85.0	63.6
$f_{\widehat{\mathbf{M}}^{(\text{WYS})}}$ (k=100)	98.7	92.1	89.2	87.9
	98.7	96.0	90.5	86.2

REFERENCES

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

Sami Abu-El-Haija, Bryan Perozzi, Amol Kapoor, Hrayr Harutyunyan, Nazanin Alipourfard, Kristina Lerman, Greg Ver Steeg, and Aram Galstyan. Mixhop: Higher-order graph convolutional architectures via sparsified neighborhood mixing. In *International Conference on Machine Learning*, ICML, 2019.

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

Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. Simple and deep graph convolutional networks. In *International Conference on Machine Learning*, ICML, 2020.

Siheng Chen, Sufeng Niu, Leman Akoglu, Jelena Kovacevic, and Christos Faloutsos. Fast, warped graph embedding: Unifying framework and one-click algorithm. *arxiv:1702.05764*, 2017.

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

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

Gene H. Golub and Charles F. Van Loan. *Matrix Computations*, pp. 257–258. John Hopkins University Press, Baltimore, 3rd edition, 1996.

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

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

William Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, NeurIPS, 2017.

Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. In *arXiv*, 2020.

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

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

C Lanczos. 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.

Omer Levy and Yoav Goldberg. Neural word embedding as implicit matrix factorization. In *Advances in Neural Information Processing Systems*, NeurIPS, pp. 2177–2185, 2014.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*, NeurIPS, pp. 3111–3119, 2013.

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

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

Jiezhong Qiu, Yuxiao Dong, Hao Ma, Jian Li, Kuansan Wang, and Jie Tang. 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.

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

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations*, ICLR, 2018.

Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks. In *International Conference on Machine Learning*, ICML, pp. 5453–5462, 2018.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, ICLR, 2019.

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

A Appendix: Implementation of Functional SVD

A.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.

A.2 TensorFlow Implementation

Since we do not explicitly calculate the $\widehat{\mathbf{M}}$ matrices displayed in Equations 6 and 10, 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 $f(\mathbf{v}) = \langle \mathbf{M}, \mathbf{v} \rangle$ for any $\mathbf{v} \in \mathbb{R}^c$ (recall: $\mathbf{M} \in \mathbb{R}^{r \times c}$)
2. Transpose[1] function t. $\forall \mathbf{v} \in \mathbb{R}^r$, $(t \circ f)(\mathbf{v}) = \langle \mathbf{M}^\top, \mathbf{v} \rangle$
3. Shape (constant) function s that knows and returns (r, c). Once transposes, should return (c, r).

A.3 Analysis

A.3.1 Norm Regularization of Wide Models

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

[1] An alternative to t can be a left-multiply function $l(\mathbf{u}) = \langle \mathbf{u}, \mathbf{M} \rangle$, however, in practice, TensorFlow is optimized for CSR matrices and computationally favors sparse-times-dense

Algorithm 1 Functional Randomized SVD, following prototype of Halko et al. (2009)

input: rank $k \in \mathbb{N}_+$, product fn $f : \mathbb{R}^c \to \mathbb{R}^r$, shape fn s, transpose fn $t : (\mathbb{R}^c \to \mathbb{R}^r) \to (\mathbb{R}^c \to \mathbb{R}^r)$
procedure fSVD(f, t, s, k)
  $(r, c) \leftarrow s()$
  $Q \sim \mathcal{N}(0,1)^{c \times 2k}$ ▷ IID Gaussian. Shape: $(c \times 2k)$
  for $i \leftarrow 1$ **to** `iterations` **do**
    $Q, \_ \leftarrow$ `tf.linalg.qr`$(f(Q))$ ▷ $(r \times 2k)$
    $Q, \_ \leftarrow$ `tf.linalg.qr`$((t \circ f)(Q))$ ▷ $(c \times 2k)$
  $Q, \_ \leftarrow$ `tf.linalg.qr`$(f(Q))$ ▷ $(r \times 2k)$
  $B \leftarrow ((t \circ f)(Q))^\top$ ▷ $(2k \times c)$
  $U, s, V^\top \leftarrow$ `tf.linalg.svd`(B)
  $U \leftarrow Q \times U$ ▷ $(r \times 2k)$
  return $U[:, :k], s[:k], V[:, :k]^\top$

Theorem 1 (Min. Norm) *If system $\widehat{\mathbf{M}}\widehat{\mathbf{W}} = \mathbf{Y}$ is underdetermined[2] with rows of $\widehat{\mathbf{M}}$ being linearly independent, then there are infinitely many solutions. Denote solution space $\widehat{\mathcal{W}}^* = \left\{ \widehat{\mathbf{W}} \;\middle|\; \widehat{\mathbf{M}}\widehat{\mathbf{W}} = \mathbf{Y} \right\}$. Then, for $k \geq$ rank$(\widehat{\mathbf{M}})$, matrix $\widehat{\mathbf{W}}^*$, defined in Eq.13 satisfies: $\widehat{\mathbf{W}}^* = \text{argmin}_{\widehat{\mathbf{W}} \in \widehat{\mathcal{W}}^*} ||\widehat{\mathbf{W}}||_F^2$*

Proof Assume $\mathbf{Y} = \mathbf{y}$ is a column vector (the proof can be generalized to matrix $\mathbf{Y}$ by repeated column-wise application[3]). SVD($\widehat{\mathbf{M}}, k$), $k \geq$ rank$(\widehat{\mathbf{M}})$, recovers the solution:

$$\widehat{\mathbf{W}}^* = \left(\widehat{\mathbf{M}}\right)^\dagger \mathbf{y} = \widehat{\mathbf{M}}^\top \left(\widehat{\mathbf{M}}\widehat{\mathbf{M}}^\top\right)^{-1} \mathbf{y}. \tag{15}$$

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

$$\widehat{\mathbf{M}}\widehat{\mathbf{W}}^* = \widehat{\mathbf{M}}\widehat{\mathbf{M}}^\top \left(\widehat{\mathbf{M}}\widehat{\mathbf{M}}^\top\right)^{-1} \mathbf{y} = \mathbf{y}.$$

Let $\widehat{\mathbf{W}}_p \in \widehat{\mathcal{W}}^*$. We must show that $||\widehat{\mathbf{W}}^*||_2 \leq ||\widehat{\mathbf{W}}_p||_2$. Since $\widehat{\mathbf{M}}\widehat{\mathbf{W}}_p = \mathbf{y}$ and $\widehat{\mathbf{M}}\widehat{\mathbf{W}}^* = \mathbf{y}$, their subtraction gives:

$$\widehat{\mathbf{M}}(\widehat{\mathbf{W}}_p - \widehat{\mathbf{W}}^*) = 0. \tag{16}$$

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

$$\begin{aligned}(\widehat{\mathbf{W}}_p - \widehat{\mathbf{W}}^*)^\top \widehat{\mathbf{W}}^* &= (\widehat{\mathbf{W}}_p - \widehat{\mathbf{W}}^*)^\top \widehat{\mathbf{M}}^\top \left(\widehat{\mathbf{M}}\widehat{\mathbf{M}}^\top\right)^{-1} \mathbf{y} \\ &= \underbrace{(\widehat{\mathbf{M}}(\widehat{\mathbf{W}}_p - \widehat{\mathbf{W}}^*))^\top}_{=0 \text{ due to Eq. 16}} \left(\widehat{\mathbf{M}}\widehat{\mathbf{M}}^\top\right)^{-1} \mathbf{y} = 0\end{aligned}$$

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

$$\begin{aligned}||\widehat{\mathbf{W}}_p||_2^2 &= ||\widehat{\mathbf{W}}^* + \widehat{\mathbf{W}}_p - \widehat{\mathbf{W}}^*||_2^2 \\ &= ||\widehat{\mathbf{W}}^*||_2^2 + ||\widehat{\mathbf{W}}_p - \widehat{\mathbf{W}}^*||_2^2 \geq ||\widehat{\mathbf{W}}^*||_2^2 \qquad \blacksquare\end{aligned}$$

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

[2] E.g., if the number of labeled examples i.e. height of $\mathbf{M}$ and $\mathbf{Y}$ is smaller than the width of $\mathbf{M}$.

[3] The minimizer for the Frobenius norm is composed, column-wise, of the minimizers $\text{argmin}_{\widehat{\mathbf{M}}\mathbf{W}_{:,j} = \mathbf{Y}_{:,j}} ||\mathbf{W}_{:,j}||_2^2$ for all j.

A.3.2 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 $f_{\widehat{\mathbf{M}}^{\text{(WYS)}}}$: Let cost of running $f_{\widehat{\mathbf{M}}^{\text{(WYS)}}} = T_{\text{mult}}$. The run-time to compute SVD, as derived in Section 1.4.2 of (Halko et al., 2009), is:

$$\mathcal{O}(kT_{\text{mult}} + (r+c)k^2). \tag{17}$$

   Since $f_{\widehat{\mathbf{M}}^{\text{(WYS)}}}$ can be defined as C (context window size) multiplications with sparse $n \times n$ matrix $\mathcal{T}$ with m non-zero entries, then running fSVD($f_{\widehat{\mathbf{M}}^{\text{(WYS)}}}, k$) costs:

$$\mathcal{O}(kmC + nk^2) \tag{18}$$

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

$$\mathcal{O}(ndL\log(k) + (n+dL)k^2 + Lmd). \tag{19}$$

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

B APPENDIX: EXPERIMENTAL DETAILS

B.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 seven graph datasets:

- Protein-Protein Interactions (PPI): a large graph where every node is a protein and an edge between two nodes indicate that the two proteins interact. Processed version of PPI was downloaded from (Grover & Leskovec, 2016).
- 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.

Table 2: Dataset Statistics

Dataset	Nodes		Edges		Source
PPI	3,852	proteins	20,881	chemical interactions	http://snap.stanford.edu/node2vec
ego-Facebook	4,039	users	88,234	friendships	http://snap.stanford.edu/data
ca-AstroPh	17,903	researchers	197,031	co-authorships	http://snap.stanford.edu/data
ca-HepTh	8,638	researchers	24,827	co-authorships	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)

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 2. In §B.2 and §B.3, unless otherwise noted, we download authors' source code from github, modify[4] 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.

B.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, $\widehat{\mathbf{L}}$ and $\widehat{\mathbf{R}}$, calculated per Equation 8: the calculation itself invokes our functional SVD (the first time) on $f_{\widehat{\mathbf{M}}^{(\text{WYS})}}$ with rank $= 32$. Hyperparameters of $f_{\widehat{\mathbf{M}}^{(\text{WYS})}}$ are λ (negative coefficient) and C (context window-size). After concatenating $\widehat{\mathbf{L}}$ and $\widehat{\mathbf{R}}$ 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. 10) and learn its parameters via rank k SVD on $f_{\widehat{\mathbf{M}}^{(\text{JKN})}}$ (the second time), as explained in §3.2. We use the validation partition to tune L, k, λ, and C.

Table 1 (left) summarizes the performance of our approach ($f_{\widehat{\mathbf{M}}^{(\text{JKN})}}$) against aforementioned baselines, showing both test accuracy and training time. While our method is competitive with state-of-the-art, it trains much faster.

B.3 ROC-AUC Link Prediction

Given a partial graph: only of 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.

[4]Modified files are in our code repo

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 $\widetilde{\text{NetMF}}$, 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 1 (right) summarizes results test ROC-AUC. For our method (denoted $f_{\widehat{\mathbf{M}}^{(\text{WYS})}}$), we call our functional SVD (Alg. 1) and pass it $f_{\widehat{\mathbf{M}}^{(\text{WYS})}}$ as defined in Eq. 14. Embeddings are set to the SVD basis (as in, §3.1) and edge score of nodes (u, v) is $\propto$ dot-product of embeddings. The last row of the table 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 $\mathbf{c} = [1, \frac{4}{5}, \frac{3}{5}, \frac{2}{5}, \frac{1}{5}]$) and C=20 for AstroPh and HepTh.

B.4 Sensitivity Analysis

While in §B.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 (left), 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 $f_{\widehat{\mathbf{M}}^{(\text{JKN})}}$. Here, we fixed $\lambda = C = 1$ and averaged 5 runs. The (tiny) error bars show the standard deviation.

Further, while in §B.3 we do SVD on $f_{\widehat{\mathbf{M}}^{(\text{WYS})}}$ with rank $k = 32$ or $k = 100$, Figure 2 (right) 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 (§A.3.1) applies only to pseudoinversion i.e. our classification models.