Watch Your Step: Learning Graph Embeddings Through Attention

Sami Abu-El-Haija
Google Research
Mountain View, CA
haija@google.com

Bryan Perozzi
Google Research
New York City, NY
bperozzi@acm.org

Rami Al-Rfou
Google Research
Mountain View, CA
rmyeid@google.com

Alex Alemi
Google Research
Mountain View, CA
alemi@google.com

ABSTRACT

Graph embedding methods represent nodes in a continuous vector space, preserving information from the graph (e.g. by sampling random walks). There are many hyper-parameters to these methods (such as random walk length) which have to be manually tuned for every graph. In this paper, we replace random walk hyper-parameters with trainable parameters that we automatically learn via backpropagation. In particular, we learn a novel attention model on the power series of the transition matrix, which guides the random walk to optimize an upstream objective. Unlike previous approaches to attention models, the method that we propose utilizes attention parameters exclusively on the data (e.g. on the random walk), and not used by the model for inference. We experiment on link prediction tasks, as we aim to produce embeddings that best-preserve the graph structure, generalizing to unseen information. We improve state-of-the-art on a comprehensive suite of real world datasets including social, collaboration, and biological networks. Adding attention to random walks can reduce the error by 20% to 45% on datasets we attempted. Further, our learned attention parameters are different for every graph, and our automatically-found values agree with the optimal choice of hyper-parameter if we manually tune existing methods.

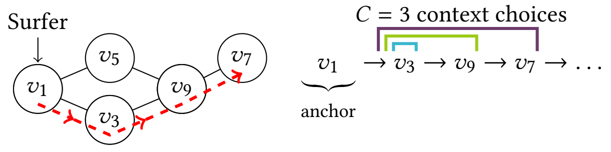
CCS CONCEPTS

• **Computing methodologies** → **Neural networks**; • **Information systems** → **Social networks**;

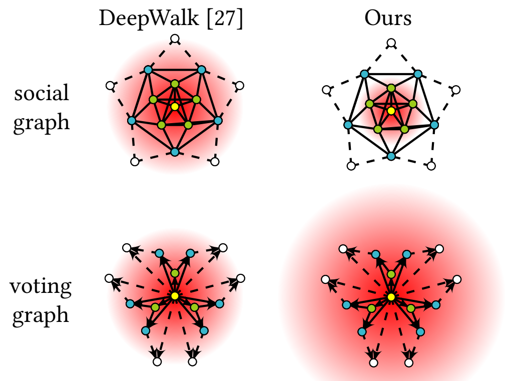
KEYWORDS

Graph, Attention, Embedding, Context Distribution

ACM Reference format:
Sami Abu-El-Haija, Bryan Perozzi, Rami Al-Rfou, and Alex Alemi. 2018. Watch Your Step: Learning Graph Embeddings Through Attention. In *Proceedings of International Workshop on Mining and Learning wit Graphs, London, UK, August 2018 (MLG'18),* 9 pages.


Surfer
v_1
v_5
v_3
v_9
v_7
C = 3 context choices
$\underbrace{v_1}_{\text{anchor}} \to v_3 \to v_9 \to v_7 \to \ldots$

(a) DeepWalk[27]: (red) paths of random walks are passed to an algorithm which iteratively selects an anchor, samples its context from fixed choices and updates embedding of anchor towards the context.



(b) Depiction of context distributions (shaded red) are assigned in earlier work (left) compared to ours (right). Top: (triadic) social graph. Bottom: (transitive) voting graph, both from the perspective of anchor node (yellow). Rather than treating them the same, Our algorithm learns left-skewed distribution the top graph and a long-tail distribution for the bottom.

Figure 1: Our contibution. We propose to learn a context distribution per graph, rather than treating all graphs the same like previous work (DeepWalk [27], node2vec[13], others).

https://doi.org/10.475/123_4

1 INTRODUCTION

Graph embedding methods based on random walks have demonstrated outstanding performance on a number of tasks including node classification [13, 27], knowledge-graph embedding [22], semi-supervised learning [33], and link prediction [2]. These methods operate by simulating many random walks on the graph and produce node embeddings based on co-occurrence statistics. However,

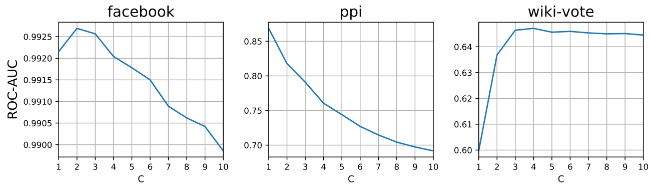


Figure 2: Motivation - Embedding methods are sensitive to hyper-parameters. Every graph prefers its own optimal C value. We plot test ROC-AUC as a function of C using node2vec [13] when embeddings are 128-dimensional. Each point is the average of 7 runs.

despite their performance gains, they can be sensitive to the particular hyper-parameters used in their training.

For example, the quality of embeddings is affected by length of the random walk and the procedure of how the context nodes are selected from around the anchor node. Hyper-parameter C controls the maximum context window size for two nodes to be considered co-visited in a random walk. Perozzi et al. [27] show that C has an impact on performance and the optimal context window size is dependent on the specific graph. This is further confirmed by Figure 2. Additionally, Levy et al. [19, their Section 3.1] show that rather than using C as a constant, assembling context of all nodes within distance C from anchor node, one should instead sample the context distance c e.g. uniformly from 1 to C as in $c \sim \mathcal{U}\{1, C\}$. Further, node2vec [13] designates two hyper-parameters for the next hop in random walks. The ratio of the two hyper-parameters determines depth versus breadth of the walk. Grover & Leskovec [13] show that different graphs prefer different hyper-parameter choices.

Implicitly, the choice of maximum window C, the context distribution $\mathcal{C}$ (e.g. $\mathcal{U}$), and random-walk hyper-parameters (e.g. node2vec's), all impose a distribution on every node's neighborhood. In general, the distrubution assigns higher weight to nearby nodes, but the specific form of the distribution is determined by the forementioned hyper-parameters. We aim to replace these hyper-parameters with trainable parameters, so that we automatically learn them per graph.

We pose graph embedding as end-to-end learning, where the random walk simulation and context sampling are replaced with a closed-form expectation over the graph transition matrix. We show mathematical equivalence between the context distribution and the co-efficients of power series of the transition matrix. We are able to learn the context distribution by learning an attention model on the power series. The attention parameters "guide" the random walk, by allowing it to focus more on short- or long-term dependencies, as best suited for the graph, while optimizing an upstream objective. In addition, we show that the optimal choice of context distribution hyper-parameters for competing methods, found by manual tuning, agrees with our automatically-found attention parameters.

Attention models have been explored in various research areas, including Natural Language Processing (NLP) [e.g. 4, 34], image recognition [24], and detecting rare events in videos [29]. We differ from those attention models in that our attention parameters are *not* part of the model. They are only part of the random walk, which generate node sequences that are then used for embedding learning. In other words, the attention parameters are not used for inference. To the best of our knowledge, this work is the first application of attention methods to random walks.

We propose two families of attention models: (1) a softmax model, (2) a geometric-decaying distribution. The former can learn arbitrary (e.g. non-monotonic) context distributions, as it has one parameter for each position in the context, while the latter learns a monotonically decaying function where the decay value is parametrized by a single trainable variable.

We evaluate our proposed attention mechanism on learning embeddings for preserving the graph structure. Strong embedding methods should be able to recover the input graph, and also infer missing edges. We evaluate on a number of challenging link prediction tasks comprised of real world datasets, including social, collaboration, and biological networks. Experiments show we substantially improve on our baselines, reducing link-prediction error by 25%-67%.

2 RELATED WORK

We review two broad classes of graph learning algorithms.

The first class is concerned with predicting labels over a graph, its edges, and/or its nodes. Typically, these algorithms process a graph (nodes and edges) as well as per-node features. These include recent graph convolution methods [e.g. 3, 6, 14, 25, 28] with spectral variants [6, 10, 15], diffusion methods [e.g. 8, 9, 11], including ones trained until fixed-point convergence [20, 30] and semi-supervised node classification [33] with low-rank approximation of convolution [17]. We differ from all these methods as (1) our algorithm is trained exclusively from the graph structure (nodes and edges), not utilizing per-node features or any labels during training, and (2) we explicitly model the relationship between all node pairs.

The second class of algorithms consist of graph embedding methods. Their primary goal is to preserve the graph structure. They explicitly model the relationship of all node pairs e.g. as dot product of node embeddings. Some methods directly use the adjacency matrix [7, 32], and others simulate random walks [2, 13, 27]. Our work falls under this class of algorithms, where inference is a scoring function $V \times V \rightarrow \mathbb{R}$, trained to score positive edges higher than negative ones. Nonetheless, we differ from existing methods as they specify the random walk and the context distribution using hyperparameters, whereas we use differentiable parameters which we jointly train while learning the embeddings, using an objective that preserves the graph structure.

3 PRELIMINARIES

3.1 Graph Embeddings

We describe Graph-Preserving embedding methods in a general framework (Eq. 1). An unweighted graph can be represented as an Adjacency Matrix $\mathbf{A} \in \{0, 1\}^{|V| \times |V|}$, where V is the set of all nodes. In general, graph embedding methods minimize an objective:

$$\min_{\mathbf{Y}} \mathcal{L}(f(\mathbf{A}), g(\mathbf{Y})); \qquad (1)$$

where $\mathbf{Y} \in \mathbb{R}^{|V| \times d}$ is a node embedding dictionary that the optimization wishes to learn, containing d dimensions per node; $f : \mathbb{R}^{|V| \times |V|} \rightarrow \mathbb{R}^{|V| \times |V|}$ is a transformation of the adjacency matrix; $g : \mathbb{R}^{|V| \times d} \rightarrow \mathbb{R}^{|V| \times |V|}$ is a pairwise edge function; and

$\mathcal{L} : \mathbb{R}^{|V|\times|V|}\times\mathbb{R}^{|V|\times|V|} \rightarrow \mathbb{R}$ is a loss function. For instance, Matrix Factorization (MF) [18] can be cast into this framework by setting the adjacency transformation to identity $f(\mathbf{A}) = \mathbf{A}$; decomposing Y into two halves, the left- and right-embedding dictionaries, $\mathbf{L} \in \mathbb{R}^{|V|\times\frac{d}{2}}$ and $\mathbf{R} \in \mathbb{R}^{|V|\times\frac{d}{2}}$, as $\mathbf{Y} = [\mathbf{L} \mid \mathbf{R}]$; setting the edge function to their outer product $g(\mathbf{Y}) = g([\mathbf{L} \mid \mathbf{R}]) = \mathbf{L} \times \mathbf{R}^T$; and setting the loss to Frobenius norm, yielding:

$$\min_{\mathbf{L},\mathbf{R}} ||\mathbf{A} - \mathbf{L} \times \mathbf{R}^T||_F \tag{2}$$

3.2 Learning Embeddings via Random Walks

Introduced by Perozzi et al.[27], this family of methods [incl. 13, 16] induce random walks along E by starting from a random node $v_0 \in sample(V)$, and repeatedly sampling an edge to transition to next node as $v_{i+1} := sample(E[v_i])$, where $E[v_i]$ are the outgoing edges from v_i. The transition sequences $(v_0, v_1, v_2, \dots)$ (i.e. random walks) are used to learn embeddings by stochastically taking every node along the sequence $v_i \in (v_0, v_1, v_2, \dots)$, and the embedding representation of this **anchor** node v_i is brought closer to the embeddings of its next neighbors, $\{v_{i+1}, v_{i+2}, \dots, v_{i+c}\}$, the **context**[1]. In practice, the context window size c can be sampled from a distribution e.g. uniform $\mathcal{U}\{1, C\}$ as explained in [19].

Let $\mathbf{D} \in \mathbb{R}^{|V|\times|V|}$ be the co-occurance matrix from random walks, with each entry D_{vu} containing the number of times nodes v and u are co-visited within context distance $c \sim \mathcal{U}\{1, C\}$, in all simulated random walks. Embedding methods utilizing random walks, can also be viewed using the framework of Eq. (1). For example, to get Node2vec [13], we can set $f(\mathbf{A}) = \mathbf{D}$, set the edge function to the embeddings outer product $g(\mathbf{Y}) = \mathbf{Y} \times \mathbf{Y}^T$, and set the loss function to negative log likelihood of softmax, yielding:

$$\min_{\mathbf{Y}} \left[\log Z - \sum_{v\in V, u\in V} D_{vu}(Y_v^T Y_u) \right], \tag{3}$$

where the normalizing constant $Z = \sum_{v,u} \exp(Y_v^T Y_u)$ can be estimated using negative sampling [13, 23].

3.2.1 Graph Likelihood. [2] learn embeddings by maximizing the **graph likelihood**:

$$\prod_{v,u\in V} \sigma(g(v, u))^{D_{vu}} (1 - \sigma(g(v, u)))^{\mathbb{1}[(v,u)\notin E]}, \tag{4}$$

where function $g : V \times V \rightarrow \mathbb{R}$ scores edges e.g. $g(v, u) = L_v^T R_u$; $\sigma(.)$ is the standard logistic, $\sigma(x) = (1 + \exp(-x))^{-1}$; and indicator function $\mathbb{1}[.]$ takes binary predicates and outputs $= 1$ if its argument is true and $= 0$ otherwise. Maximizing the graph likelihood pushes $\sigma(g(v, u))$ towards 1 if value D_{vu} is large and pushes it towards 0 if $(v, u) \notin E$.

Using our matrix notation, we write our main objective: the Negative Log Graph Likelihood (NLGL), as:

$$\left\| -\mathbf{D} \circ \log\left(\sigma(\mathbf{L} \times \mathbf{R}^T)\right) - \mathbb{1}[\mathbf{A} = 0] \circ \log\left(1 - \sigma(\mathbf{L} \times \mathbf{R}^T)\right) \right\|_1, \tag{5}$$

which we minimize w.r.t model embedding parameters $\mathbf{L}, \mathbf{R} \in \mathbb{R}^{|V|\times\frac{d}{2}}$, where logistic $\sigma(x)$ and indicator $\mathbb{1}[.]$ functions applied element-wise; $\circ$ is the hadamard product; and the L1-norm $||.||_1$ of a matrix is the sum of its entries. The entries of this matrix are positive because $0 < \sigma(.) < 1$.

4 METHOD

following our general framework (Eq 1), we set $g(\mathbf{Y}) = g([\mathbf{L} \mid \mathbf{R}]) = \mathbf{L}\times\mathbf{R}^T$ and model $f(\mathbf{A}) = \mathbb{E}[\mathbf{D}]$, as an expectation on co-occurrence matrix produced from simulated random walk. Using this closed form, we introduce attention parameters on the random walk, which live in an extended version of NLGL (Eq. 5).

4.1 Walk Co-occurance Expectation $\mathbb{E}[\mathbf{D}]$

Let $\mathcal{T}$ be the transition matrix for a graph, which can be calculated by normalizing the rows of its adjacency matrix $\mathbf{A}$ to one[2]. Given an initial probability distribution $p^{(0)} \in \mathbb{R}^{|V|}$ of a random surfer, it is possible to find the distribution of the surfer after one step conditioned on $p^{(0)}$ as $p^{(1)} = p^{(0)^T}\mathcal{T}$ and after k steps as $p^{(k)} = p^{(0)^T}(\mathcal{T})^k$, where $(\mathcal{T})^k$ multiplies matrix $\mathcal{T}$ with itself k-times. We are interested in an analytical expression for $\mathbb{E}[\mathbf{D}]$, the expectation over co-occurrence matrix produced by simulated random walks. A closed form expression for this matrix will allow us to perform end-to-end learning.

In practice, random walk methods based on DeepWalk [27] do not use C as a hard limit; instead, given walk sequence $(v_1, v_2, \dots)$, they sample $c \sim \mathcal{U}\{1, C\}$ separately for each anchor node v_i and use context nodes $(v_{i+1}, v_{i+2}, \dots, v_{i+c})$, that are within c-steps away from anchor node v_i. In expectation, nodes $v_{i+1}, v_{i+2}, v_{i+3}, \dots$, will appear as context for anchor node v_i, respectively with probabilities $1, 1 - \frac{1}{C}, 1 - \frac{2}{C}, \dots$. We can write an expectation on $\mathbf{D}$:

$$\mathbb{E}\left[\mathbf{D}^{\textsc{DeepWalk}[C]}\right] = \sum_{k=1}^{C} \Pr(c \geq k)\tilde{\mathbf{P}}^{(0)}(\mathcal{T})^k, \tag{6}$$

where $\mathbf{D}^{\textsc{DeepWalk}[C]}$ is the co-occurrence matrix, if DeepWalk simulated random walks with maximum context window C; $\Pr(c \geq k)$ indicates the probability of node with distance k from anchor to be selected; and $\tilde{\mathbf{P}}^{(0)} \in \mathbb{R}^{|V|\times|V|}$ is the initial starting positions matrix, described in next subsection.

Since $\Pr(c = k) = \frac{1}{C}$ for all $k = \{1, 2, \dots, C\}$, we can expand $\Pr(c \geq k) = \sum_{j=k}^{C} P(c = j)$, and re-write the expectation as:

$$\mathbb{E}\left[\mathbf{D}^{\textsc{DeepWalk}[C]}\right] = \tilde{\mathbf{P}}^{(0)} \sum_{k=1}^{C} \left[1 - \frac{k-1}{C}\right] (\mathcal{T})^k. \tag{7}$$

Eq. (7) is derived, step-by-step, in the Appendix. The term $\left[1 - \frac{k-1}{C}\right]$ imposes a linear decay on the distribution of context nodes, with a decay factor of $\frac{1}{C}$. Therefore, given a random walk sequence $(v_0, v_1, \cdots)$, the embedding for each node v_i along the sequence, in expectation, is updated *closer* to the convex combination of $v_{i+1} + (1 - \frac{1}{C}) \times v_{i+2} + (1 - \frac{2}{C}) \times v_{i+3} + \cdots + \frac{1}{C} \times v_{i+C}$.

As an aside, we note that this word2vec-style linear decay in Eq. (7) is not the only possible choice. Another example comes from

[1] Our definition of context differs [27] and [13]. We use context nodes that strictly *follow* their anchor, whereas those methods use define their context that *surrounds* the anchor, like $\{v_{i-c}, \dots, v_{i-2}, v_{i-1}, v_{i+1}, v_{i+2}, \dots, v_{i+c}\}$. Our definition preserves the direction of edges.

[2] We use *right stochastic* transition. See https://en.wikipedia.org/wiki/Stochastic_matrix

GloVe [26], which uses the harmonic series. Using our notation, we can write the expectation on $\mathbf{D}$, obtained by GloVe's decay, as:

$$\mathbb{E}\left[\mathbf{D}^{\text{GloVe}[C]}\right] = \tilde{\mathbf{P}}^{(0)} \sum_{k=1}^{C} \frac{1}{k} (\mathcal{T})^k . \tag{8}$$

4.1.1 Choice of $\tilde{\mathbf{P}}^{(0)}$*.* Random walk simulation of node2vec [12] starts 80 walks from every graph node $v \in V$. Therefore, in our experiments, we set $\tilde{\mathbf{P}}^{(0)} := \text{diag}(80, 80, \dots, 80)$. This initial condition yields D_{vu} to be the expected number of times that u is visited if we started 80 walks from v. There can be other reasonable choices. Nonetheless, we use what worked well in practice for [13, 27]. We leave the search for a better $\tilde{\mathbf{P}}^{(0)}$ as future work.

4.2 Parametrizing the Power Series of the Transition Matrix

We propose to parameterize the importance of different terms in the power series of the transition matrix. Instead of pre-determining the coefficient to each $(\mathcal{T})^k$, our coefficients are learnable and come from probability distribution $Q = (Q_1, Q_2, \cdots, Q_C)$ with $Q_k \geq 0$ and $\sum_k Q_k = 1$, assigning weight Q_k to $(\mathcal{T})^k$. Formally, we propose the parametrized conditional expectation:

$$\mathbb{E}[\mathbf{D} \mid Q_1, Q_2, \dots Q_C] = \tilde{\mathbf{P}}^{(0)} \sum_{k=1}^{C} Q_k (\mathcal{T})^k . \tag{9}$$

Existing random walk methods can be viewed as using a fixed distribution for Q, which has been hand engineered by their designers. In the case of DeepWalk, $Q_k = \left[1 - \frac{k-1}{C}\right]$. Correspondingly for GloVe, $Q_k = \frac{1}{k}$.

4.3 Attention Models on Random Walks

Rather than using an engineered distribution Q, we are interested in estimating Q automatically, via backpropagation. To that end, we propose two attention models, which guide the random surfer on "where to attend to" as a function of distance from the source node.

4.3.1 Softmax Attention. We first propose modeling the context distribution Q as output of softmax:

$$(Q_1, Q_2, Q_3, \dots) = \text{softmax}((q_1, q_2, q_3, \dots)), \tag{10}$$

where the variables q_k are trained via backpropagation. Our hypothesis is as follows. If we don't impose a specific formula on $Q = (Q_1, Q_2, \dots Q_C)$, other than regularized softmax, then we can use very large values of C and allow every graph to learn its own form of Q with its preferred sparsity and own decay form. Should the graph structure require a small C, then the optimization would discover a left-skewed Q with all of probability mass on $\{Q_1, Q_2\}$ and $\sum_{k>2} Q_k \approx 0$. However, if according to the objective, a graph is better preserved by making longer walks, then they can learn to use a large C (e.g. using uniform or even right-skewed Q distribution), focusing more attention on longer distance connections in the random walk.

To this end, we propose to train softmax attention model on the infinite power series of the transition matrix. We define an expectation on our proposed random walk matrix $\mathbf{D}^{\text{softmax}[\infty]}$ as[3]:

$$\mathbb{E}\left[\mathbf{D}^{\text{softmax}[\infty]} \,\middle|\, q_1, q_2, q_3, \dots\right] = \tilde{\mathbf{P}}^{(0)} \lim_{C \to \infty} \sum_{j=1}^{C} \frac{1}{e^{q_j}} \sum_{k=1}^{C} e^{q_k} (\mathcal{T})^k , \tag{11}$$

where $q_1, q_2, \dots$ are jointly trained with the embeddings to minimize our objective. To reduce notation, we also refer to the expectation (Eq. 11) as $\mathbf{D}^{\text{softmax}[\infty]}$.

4.3.2 λ Geometric Decay Attention. We propose a second attention mechanism. Contrary to the softmax attention, which assumes that every step has its own (learnable) importance, our λ-decay attention assumes that coefficients of $(\mathcal{T})^k$ must geometrically decay with k. Specifically, we assume that vertices closer to anchor nodes in random walks are more important than further ones. We propose context distribution Q:

$$Q_k = \frac{\sigma(\lambda)^k}{\sum_{j=1}^{C} \sigma(\lambda)^j}, \tag{12}$$

where the logistic[4] output range is in $[0, 1]$. This yields a conditional expectation on $\mathbf{D}$, parametrized with λ:

$$\mathbb{E}\left[\mathbf{D}^{\lambda\text{-decay}[C]} \,\middle|\, \lambda\right] = \tilde{\mathbf{P}}^{(0)} \frac{1}{\sum_{j=1}^{C} \sigma(\lambda)^j} \sum_{k=1}^{C} \sigma(\lambda)^k (\mathcal{T})^k . \tag{13}$$

This allows tuning the attention of a random walker to specific distances for each graph, relying only on a single learned parameter. We compare both attention mechanisms in the Section 5.

4.4 Training Objective

The final training objective for the Softmax attention mechanism, coming from the NLGL Eq. (5),

$$\min_{\mathbf{L}, \mathbf{R}, \mathbf{q}} \beta ||\mathbf{q}||_2^2 + \Big\| -\mathbb{E}[\mathbf{D} \mid \mathbf{q}] \circ \log\left(\sigma(\mathbf{L} \times \mathbf{R}^T)\right) - \mathbb{1}[\mathbf{A} = 0] \circ \log\left(1 - \sigma(\mathbf{L} \times \mathbf{R}^T)\right) \Big\|_1$$

is minimized w.r.t attention parameter vector $\mathbf{q} = (q_1, q_2, \dots)$ and node embeddings $\mathbf{L}, \mathbf{R} \in \mathbb{R}^{|V| \times \frac{d}{2}}$. Hyper-parameter $\beta \in \mathbb{R}$ applies L2 regularization on the attention parameters. We emphasize that our attention parameters $\mathbf{q}$ live within the expectation over data $\mathbf{D}$, and are not part of the model $(\mathbf{L}, \mathbf{R})$ and are not required for inference. The constraint $\sum_k Q_k = 1$, through the softmax activation, prevents $\mathbb{E}[\mathbf{D}^{\text{softmax}}]$ from collapsing into a zero matrix. The objective for the λ-decay attention model is similar, except that we don't regularize the λ parameter.

5 EXPERIMENTS

5.1 Link Prediction Experiments

We evaluate the quality of embeddings produced when random walks are augmented with attention, through experiments on link

[3]We do *not* actually unroll the summation in Eq. (11) an infinite number of times. Our experiments show that unrolling it 10 or 20 times is sufficient to obtain state-of-the-art results.

[4]In practice, rather than setting $Q \propto \sigma(\lambda)^k$, we set $Q \propto 0.99\sigma(\lambda)^k + 0.01$, to avoid numerical errors.

Dataset	$\|V\|$	$\|E\|$	nodes	edges	directed?
wiki-vote	7,066	103,663	users	votes	yes
ego-Facebook	4,039	88,234	users	friendship	no
ca-AstroPh	17,903	197,031	researchers	co-authorship	no
ca-HepTh	8,638	24,827	researchers	co-authorship	no
PPI [31]	3,852	20,881	proteins	chemical interaction	no

Table 1: Datasets used in our experiments.

Dataset	dim	Adjacency Matrix			**D** by Simulation			Attention Walks (*ours*)		Error Reduction
		Eigen Maps	SVD	DNGR	node2vec $C = 2$	node2vec $C = 5$	Asym Proj	λ-decay (13)	softmax (11)	
wiki-vote	64	61.3	86.0	59.8	64.4	63.6	91.7	**93.5 ± 0.62**	**93.8 ± 0.13**	25.2%
	128	62.2	80.8	55.4	63.7	64.6	91.7	92.9 ± 0.73	**93.8 ± 0.05**	25.2%
ego-Facebook	64	96.4	96.7	98.1	99.1	99.0	97.4	**99.3 ± 0.02**	**99.4 ± 0.10**	33.3%
	128	95.4	94.5	98.4	99.3	99.2	97.3	99.3 ± 0.03	**99.5 ± 0.03**	28.6%
ca-AstroPh	64	82.4	91.1	93.9	97.4	96.9	95.7	**98.6 ± 0.03**	97.9 ± 0.21	46.2%
	128	82.9	92.4	96.8	97.7	97.5	95.7	**98.6 ± 0.03**	98.1 ± 0.49	39.1%
ca-HepTh	64	80.2	79.3	86.8	90.6	91.8	90.3	91.4 ± 0.17	**93.6 ± 0.06**	22.0%
	128	81.2	78.0	89.7	90.1	92.0	90.3	92.2 ± 0.18	**93.9 ± 0.05**	23.8%
PPI	64	70.7	75.4	76.7	79.7	70.6	82.4	**90.0 ± 0.03**	**89.8 ± 1.05**	43.5%
	128	73.7	71.2	76.9	81.8	74.4	83.9	90.4 ± 0.06	**91.0 ± 0.28**	44.2%

Table 2: Results on Link Prediction Evaluation. Shown is the ROC-AUC. Note that both our proposed attention models perform well on the task, and substantially better than all baselines. Error Reduction is calculated as $\frac{(1-\text{them})-(1-\text{us})}{(1-\text{them})}$, where "them" the best performer from prior methods and "us" is the best performer of the attention mechanisms.

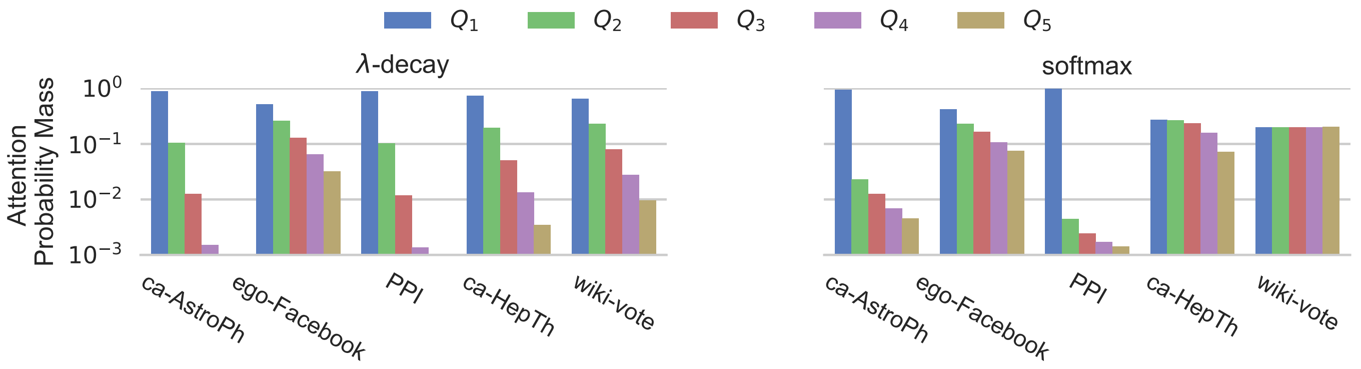


Figure 3: Log-scale of learned Attention weights Q across datasets for both λ-decay (left) and softmax (right). Auotmatically-found attention agrees with manual tuning node2vec on hyper-parameter C, as shown in Figure 2. We note that increasing β will tend to push the distribution towards Uniform and less spiky.

prediction [21]. Link prediction is a challenging task, with many real world applications in information retrieval, recommendation systems and social networks. As such, it has been used to study the properties of graph embeddings [13, 27]. Such an intrinsic evaluation emphasizes the structure-preserving properties of embedding.

Our experimental setup is designed to determine how well the embeddings produced by a method captures the topology of the graph. We measure this in the manner of [13]: remove a fraction (=50%) of graph edges, learn embeddings from the remaining edges, and measure how well the embeddings can recover those edges which have been removed. More formally, we split the graph edges E into two partitions of equal size E_{train} and E_{test} such that the training graph is connected. We also sample non existent edges ($(u, v) \notin E$) to make E^-_{train} and E^-_{test}. We use (E_{train}, E^-_{train}) for training and model selection, and use (E_{test}, E^-_{test}) to compute evaluation metrics. We train our models using TensorFlow, with PercentDelta optimizer [1]. For the results Table 2, we use $\beta = 0.5$ and we tried various values for C in $\{5, 10, 20, 30\}$, and the results made no difference, confirming that our method effectively uses a portion of the context distribution. To ensure repeatability of results, we release our evaluation scripts[5].

Datasets: Table 1 describes the datasets used in our experiments. Datasets available from SNAP https://snap.stanford.edu/data.

Baselines: We evaluate against many baselines. For all methods, we calculate $g(\mathbf{Y}) \in \mathbb{R}^{|V| \times |V|}$, and extract entries from $g(\mathbf{Y})$

[5]To be uploaded to github, by camera-ready submission, if paper gets accepted.

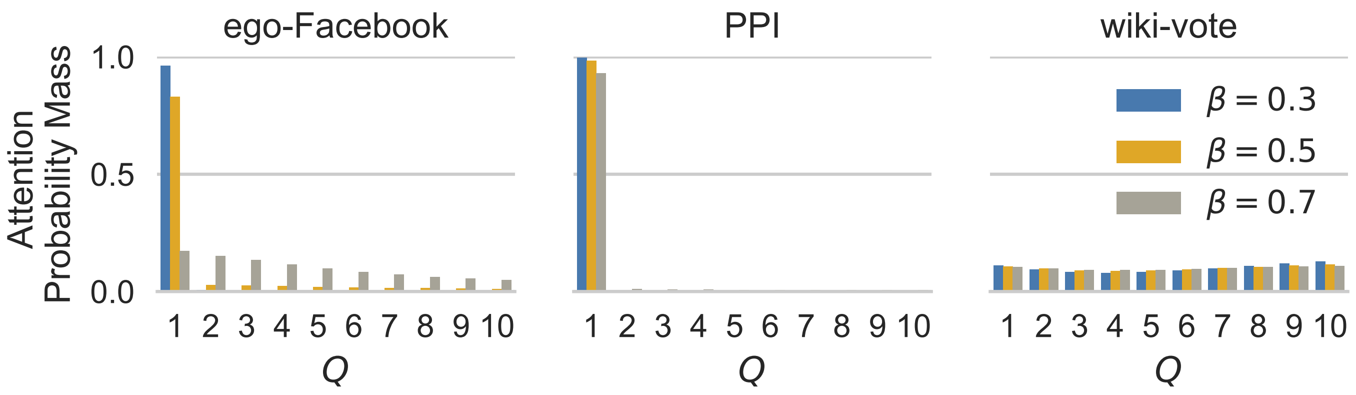


Figure 4: Affect of varying the regularization β of the softmax attention mechanism when $C = 10$, drawn in linear scale. Note that distributions can quickly tail off to zero (ego-Facebook and PPI), but some graphs prefer to use the whole space (wiki-vote).

corresponding to positive and negative test edges, then use them to compute ROC AUC. We compare against following baselines:

– **EigenMaps** [5]. Uses $\mathbf{A}$ and minimizes Euclidean distance of adjacent nodes embedding vectors. We use sk-learn's eigendecomposition. Inference is through $g(\mathbf{Y})_{uv} = \exp(-||Y_u - Y_v||^2)$.

– **SVD**. Singular value decomposition of $\mathbf{A}$. Eq. (2) with added orthonormality constraints. We use scikit-learn's SVD decomposition. $g(\mathbf{Y}) = g\left([\mathbf{L} \mid \mathbf{R}]\right) = \mathbf{L} \times \mathbf{R}^T$.

– **DNGR** [7]. Non-linear (i.e. deep) embedding of nodes, using an auto-encoder on $\mathbf{A}$. We use author's code to learn the deep embeddings $\mathbf{Y}$ and use for inference $g(\mathbf{Y}) = \mathbf{Y}\mathbf{Y}^T$.

– **node2vec** [13]. Simulates random walks and uses word2vec to learn node embeddings. Minimizes objective in Eq. (3). For Table 2, we use author's code to learn embeddings $\mathbf{Y}$ then use $g(\mathbf{Y}) = \mathbf{Y}\mathbf{Y}^T$. We run with $C = 2$ and $C = 5$. We sweep C in Figure 2, showing indeed that there are no good default for C that works best across datasets.

– **AsymProj** [2]. Learns edges as asymmetric projections in a deep embedding space, trained by maximizing the graph likelihood (Eq. 4). We take results from authors.

Results: Our results, summarized in Table 2, show that our proposed methods substantially outperform all baseline methods. Specifically, we see that the error is reduced by up to 45% over baseline methods which have fixed context definitions. This shows that by parameterizing the context distribution and allowing each graph to learn its own distribution, we can better preserve the graph structure (and thereby better predict missing edges).

Discussion: Both attention models frequently perform quite similarly, despite the fact that the softmax model has more flexibility over its distribution. This implies that in many cases, the geometric decay assumption over the context holds. Interestingly, the two datasets where the attention models differ the most (ca-AstroPh and ca-HepTh) are the most similar to each other (both are collaboration networks of researchers). This illustrates the importance of the attention models we propose – seemingly similar networks may require drastically different context distributions in order to best preserve their individual graph structure.

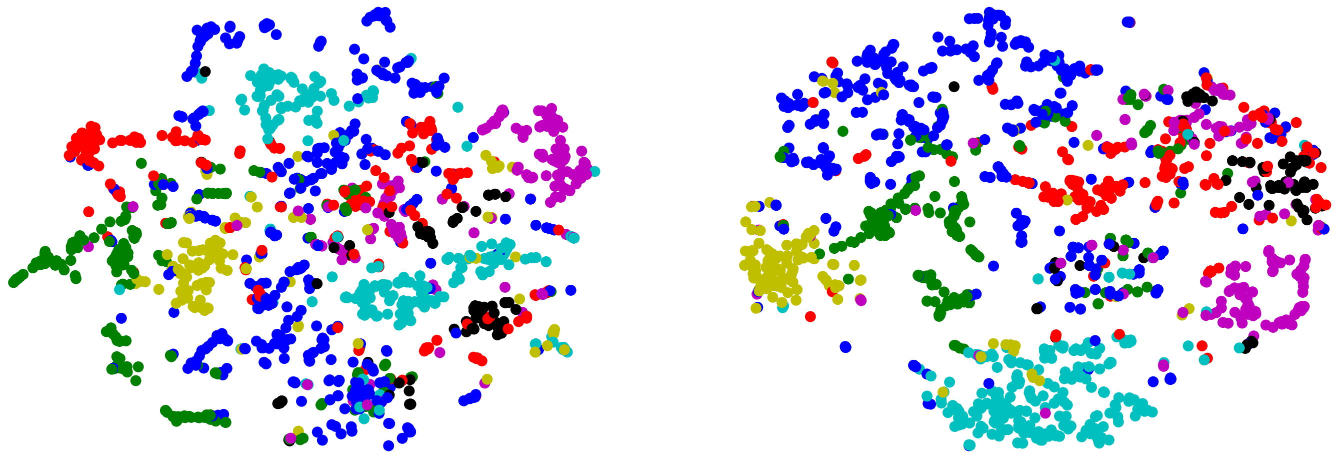
Figure 3 shows how the learned attention weights Q vary across datasets for both the softmax model and the λ-decay model. Each dataset learns its own attention form, and they look similar for the two attention mechanisms except that the λ has less degrees of freedom. The hyper-parameter C determines the highest power of the transition matrix, and hence the maximum context size available to the attention model. We suggest using large values for C, since the attention weights can effectively use a subset of the transition matrix powers. For example, if a network needs only 2 hops to be accurately represented, then it is possible for the softmax attention model to learn $Q_3, Q_4, \cdots \approx 0$. Figure 4 shows how varying the regularization term β allows the softmax attention model to "attend to" only what each dataset requires. We observe that for most graphs, the majority of the mass gets assigned to Q_1, Q_2. This shows that shorter walks are more beneficial for most graphs. However, on wiki-vote, better embeddings are produced by paying attention to longer walks, as its softmax Q is uniform-like, with a slight right-skew.

5.2 Sensitivity Analysis

As we remove context hyper-parameters, maximum window size and form of distribution: C and e.g. $\mathcal{U}$, we introduce other hyper-parameters, specifically walk length (also C) and regularizer β for the softmax attention model. Nonetheless, we show that our method is robust to various choices of these two. Figures 3 (left) and 4 show that the softmax attention weights drop to almost zero if the graph can be preserved using shorter walks, which is not possible with fixed-form distributions (e.g. $\mathcal{U}$). In addition, we show the model test accuracy when sweeping our hyper-parameters C and β in Figure 6. We observe that all the accuracy metrics are within 1% to 2%, when varying these hyper-parameters, and are all still well-above our baselines which sample from a fixed-form context distribution.

5.3 Node Classification Experiments

We conduct node classification experiments, on two citation datasets, Cora and Citeseer, with the following statistics: Cora contains (2, 708 nodes, 5, 429 edges and $K = 7$ classes); and Citeseer contains (3, 327 nodes, 4, 732 edges and $K = 6$ classes). We learn embeddings from only the graph structure (nodes and edges), without observing node features nor labels during training. Figure 5 shows t-SNE visualization of the Cora dataset, comparing our method



(a) node2vec **(b) Ours (softmax attention)**

Figure 5: t-SNE visualization of node embeddings for Cora dataset. We run node2vec and our softmax model, and both do not accept labels during training. However, the labels are used to color nodes. Both algorithms are doing a decent job in discovering clusters that correspond to node labels. However, qualitatively, our embeddings acheives better separation.

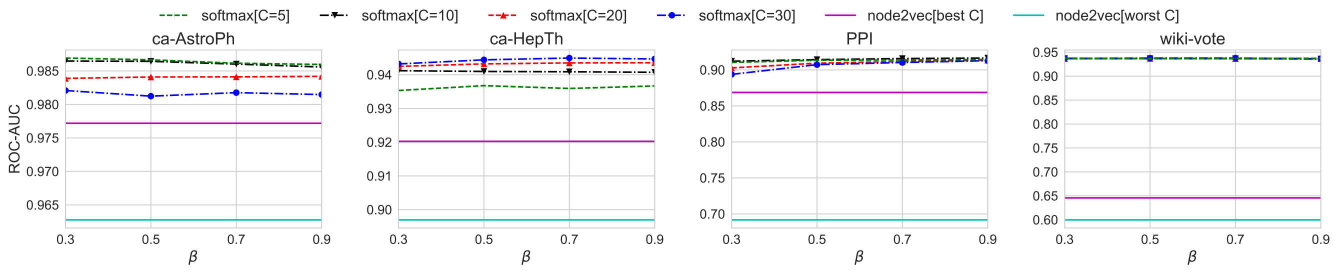


Figure 6: Sensitivity Analysis of softmax attention model. We show 4 (dashed) lines, each for a choice of C, plotting test accuracy against regularization parameter β. For reference, we show the best and worst node2vec performer for $C \in [1, 10]$, detailed in Figure 2 (here lines are horizontal as they do not depend on β). Our method is robust to reasonable choices of these hyper-parameters. All experiments trained a 128-dim embedding, $\mathrm{Y} \in \mathbb{R}^{|V| \times 128}$.

with node2vec [13]. For classification, we follow the data splits of [33], with only 140 and 120 nodes for training, respectively, for the Cora and Citeseer dataset. Test partitions include 1000 nodes. We predict labels $\widetilde{L} \in \mathbb{R}^{|V| \times K}$ as:

$$\widetilde{L} = \exp\left(\alpha g(\mathrm{Y})\right) \times L_{\text{train}}, \tag{14}$$

where binary $L_{\text{train}} \in \{0, 1\}^{|V| \times K}$ contains rows of ones corresponding to nodes in training set and zeros elsewhere. The scalar $\alpha \in \mathbb{R}$ is manually tuned on the validation set. The classification results, summarized in Table 3, show that our model learns a better unsupervised representation than previous methods, that can then be used for supervised tasks. We do not compare against other methods that utilize node features during training and inference [incl. 17, 33], as our method is able to predict labels given only the graph structure.

Our classification prediciton function (Eq. 14) contains only one scalar parameter α. It can be thought of a "smooth" k-nearest-neighbors, as it takes a weighted average of known labels, where the weights are exponential of the dot-product similarity.

Dataset	DeepWalk	node2vec $C = 5$	softmax (ours)
Cora	67.2	63.1	**67.9**
Citeseer	43.2	45.6	**51.5**

Table 3: Classification accuracy for two citation datasets. Results for DeepWalk are copied from [17]. We generated results for node2vec and ours, using Equation 14

6 CONCLUSION

In this paper, we propose an attention mechanism for learning the context distribution used in graph embedding methods. We derive the closed-form expectation of DeepWalk [27] co-occurance statistics, showing an equivalence between the context distribution hyper-parameters, and the co-efficients of the power series of the graph transition matrix. Then, we propose to replace the context hyper-parameters with trainable models, that we learn jointly with the embeddings on an objective that preserves the graph structure

(Negative Log Graph Likelihood, NLGL). Specifically, we propose two attention models: softmax and geometric decay. The first allows learning a free-form contexts distribution with a parameter per step, and the second imposes a geometric decay as a function of distance from anchor with one model parameter: the decay rate.

We show significant improvements on link prediction and node classification over state-of-the-art baselines (that use a fixed-form context distribution), reducing error on link prediction and classification, respectively by up to 40% and 10%. In addition to improved performance (by learning distributions of arbitrary forms), our method can obviate the manual grid search over hyper-parameters: walk length and form of context distribution, which can drastically fluctuate the quality of the learned embeddings and are different for every graph. On the datasets we consider, we show that our method is robust to its hyper-parameters, as described in Section 5.2. Our visualizations of converged attention weights convey to us that some graphs (e.g. voting graphs) can be better preserved by using longer walks, while other graphs (e.g. protein graphs) contain more information in short dependencies and require shorter walks.

We believe that our contribution in replacing these sampling hyperparameters with a learnable context distribution is general and can be applied to many domains and modeling techniques in graph representation learning.

REFERENCES

[1] Sami Abu-El-Haija. 2017. Proportionate gradient updates with PercentDelta. In *arXiv*.

[2] Sami Abu-El-Haija, Bryan Perozzi, and Rami Al-Rfou. 2017. Learning Edge Representations via Low-Rank Asymmetric Projections. In *ACM International Conference on Information and Knowledge Management (CIKM)*.

[3] James Atwood and Don Towsley. 2016. Diffusion-Convolutional Neural Networks. In *Advances in Neural Information Processing Systems (NIPS)*.

[4] Dzmitry Bahdanau, KyungHyun Cho, and Yoshua Bengio. 2015. Neural Machine Translation by Jointly Learning to Align and Translate. In *International Conference on Learning Representations (ICLR)*.

[5] Mikhail Belkin and Partha Niyogi. 2003. Laplacian Eigenmaps for Dimensionality Reduction and Data Representation. In *Neural Computation*.

[6] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. 2013. Spectral networks and deep locally connected networks on graphs. In *International Conference on Learning Representations*.

[7] Shaosheng Cao, Wei Lu, and Qiongkai Xu. 2016. Deep Neural Networks for Learning Graph Representations. In *Proceedings of the Association for the Advancement of Artificial Intelligence*.

[8] Liang-Chieh Chen, Alexander Schwing, Alan Yuille, and Raquel Urtasun. 2015. Learning Deep Structured Models. In *International Conference on Machine Learning*.

[9] Hanjun Dai, Bo Dai, and Le Song. 2016. Discriminative Embeddings of Latent Variable Models for Structured Data. In *International Conference on Machine Learning*.

[10] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering. In *Advances in Neural Information Processing Systems (NIPS)*.

[11] D. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik, and R. Adams. 2015. Convolutional Networks on Graphs for Learning Molecular Fingerprints. In *Advances in Neural Information Processing Systems (NIPS)*.

[12] Aditya Grover. 2016. *GitHub: node2vec code*. Technical Report. https://github.com/aditya-grover/node2vec

[13] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable Feature Learning for Networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

[14] W. Hamilton, R. Ying, and J. Leskovec. 2017. Inductive Representation Learning on Large Graphs. In *NIPS*.

[15] Mikael Henaff, Joan Bruna, and Yann LeCun. 2015. Deep Convolutional Networks on Graph-Structured Data. In *arXiv:1506.05163*.

[16] Ganesh J, Soumyajit Ganguly, Manish Gupta, Vasudeva Varma, and Vikram Pudi. 2016. Author2Vec: Learning Author Representations by Combining Content and Link Information. In *Proceedings of the International Conference Companion on World Wide Web (WWW) (WWW '16 Companion)*.

[17] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *International Conference on Learning Representations (ICLR)*.

[18] Y. Koren, R. M. Bell, and C. Volinsky. 2009. Matrix factorization techniques for recommender systems. In *IEEE Computer*.

[19] Omer Levy, Yoav Goldberg, and Ido Dagan. 2015. Improving Distributional Similarity with Lessons Learned from Word Embeddings. In *TACL*.

[20] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel. 2016. Gated Graph Sequence Neural Networks. In *International Conference on Learning Representations*.

[21] D. Liben-Nowell and J. Kleinberg. 2007. The Link-Prediction Problem for Social Networks. In *Journal of American Society for Information Science and Technology*.

[22] Y. Luo, Q. Wang, B. Wang, and L. Guo. 2015. Context-Dependent Knowledge Graph Embedding. In *Conference on Emperical Methods in Natural Language Processing (EMNLP)*.

[23] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeff Dean. 2013. Distributed Representations of Words and Phrases and their Compositionality. In *Advances in Neural Information Processing Systems NIPS*.

[24] Volodymyr Mnih, Nicolas Heess, Alex Graves, and koray kavukcuoglu. 2014. Recurrent Models of Visual Attention. In *Advances in Neural Information Processing Systems (NIPS)*.

[25] M. Niepert, M. Ahmed, and K. Kutzkov. 2016. Learning Convolutional Neural Networks for Graphs. In *International Conference on Machine Learning (ICML)*.

[26] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global Vectors for Word Representation. In *Conference on Empirical Methods in Natural Language Processing, EMNLP*.

[27] B. Perozzi, R. Al-Rfou, and S. Skiena. 2014. DeepWalk: Online Learning of Social Representations. In *Knowledge Discovery and Data Mining (KDD)*.

[28] Arantxa Casanova Adriana Romero Pietro LiÃš Yoshua Bengio Petar VeliÄŊkoviÄĞ, Guillem Cucurull. 2018. Graph Attention Networks. In *International Conference on Learning Representations (ICLR)*.

[29] Vignesh Ramanathan, Jonathan Huang, Sami Abu-El-Haija, Alexander Gorban, Kevin Murphy, and Li Fei-Fei. 2016. Detecting Events and Key Actors in Multi-Person Videos. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[30] F. Scarselli, M. Gori, A.C. Tsoi, M. Hagenbuchner, and G. Monfardini. 2009. The Graph Neural Network Model. In *IEEE Trans. on Neural Networks*.

[31] C. Stark, B.J. Breitkreutz, T. Reguly, L. Boucher, A. Breitkreutz, and M. Tyers. 2006. BioGRID: A General Repository for Interaction Datasets. In *Nucleic Acids Research*. https://www.ncbi.nlm.nih.gov/pubmed/16381927

[32] D. Wang, P. Cui, and W. Zhu. 2016. Structural Deep Network Embedding. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

[33] Z. Yang, W. Cohen, and Ruslan Salakhutdinov. 2016. Revisiting Semi-Supervised Learning with Graph Embeddings. In *International Conference on Machine Learning (ICML)*.

[34] Zichao Yang, Diyi Yang1, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. 2016. Hierarchical Attention Networks for Document Classification. In *Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*.

7 APPENDIX

7.1 Step-by-step Derivation of Equation (7)

Let $x(k)$ be the position of random surfer at time k. Speficically, $x : \mathbb{Z}^+ \longrightarrow V$. We assume a Markov chain: The value of $x(k)$ only depends on previous step: $x(k-1)$. To calulate the expectation $\mathbb{E}[\mathbf{D}]$, the square node-to-node co-occurence matrix, we start by calculating one entry at a time: $\mathbb{E}[D_{uv}]$, the expected number of times that u is selected in v's context. Let $W_v(k)$ be the context that gets sampled if v is visited at the k^{th} step. Concretely, if $x(k) = v$, and the random walker continues the sequence, $x(k+1) = v'_1 \rightarrow x(k+2) = v'_2 \rightarrow x(k+3) = v'_3 \dots$, then the context set of DeepWalk can be defined as $W_v(k) = \{v_j \mid j \geq c\}$, where $c \sim \mathcal{U}\{1, C\}$ We would like to count the event $u \in W_v(k)$ for every $k \in \{1, 2, \dots, C\}$.

Using Markov Chain, we can write:

$$\Pr(x(i+k) = u \mid x(i) = v) = \Pr(x(k) = u \mid x(0) = v) = \left(\mathcal{T}^k\right)_{uv} \tag{15}$$

Now, if node u was visited k steps after node v, then the probabilitiy of it being sampled is given by:

$$\Pr(u \in W_v(k) \mid x(k) = u, x(0) = v). \tag{16}$$

In case of DeepWalk [27], probability above equals:

$$\Pr(k \leq c \mid x(k) = u, x(0) = v) \text{ where } c \sim \mathcal{U}\{1, C\}, \tag{17}$$

and event $k \leq c$ is independant of the condition $(x(k) = u \cap x(0) = v)$. Further, event $k \leq c$ can be partitioned and Eq. (17) can be written as

$$\Pr(c = k \cup c = k+1 \cup \cdots \cup c = C) \tag{18}$$

$$= \sum_{j=k}^{C} \Pr(c = j) \tag{19}$$

$$= (C - k + 1)\left(\frac{1}{C}\right) = 1 - \frac{k-1}{C}, \tag{20}$$

where second line is trivial since the events $c = j$ are disjoint. We can now use Bayes' rule to derive the probability of u being visited k steps after v *<u>and</u>* being selected in v's sampled context, as:

$$\begin{aligned} &\Pr(u \in W_v(k), x(k) = u \mid x(0) = v) \\ &= \Pr(u \in W_v(k) \mid x(k) = u, x(0) = v) \Pr(x(k) = u \mid x(0) = v) \\ &= \left(1 - \frac{k-1}{C}\right)\left(\mathcal{T}^k\right)_{uv} \end{aligned} \tag{21}$$

Now, let E_{vku} be the event that a walker visits v and after k steps, visits u and selects it part of its context. This event happens with the probability indicated in Equation 21. Concretely,

$$\mathbb{E}\left[E_{vku} \mid x(0) = v\right] = \left(1 - \frac{k-1}{C}\right)\left(\mathcal{T}^k\right)_{uv}. \tag{22}$$

Let E_{v*u} count the events $\{E_{vku} : k \in [1, C]\}$, then:

$$\mathbb{E}\left[E_{v*u} \mid x(0) = v\right] = \mathbb{E}\left[\sum_{k=1}^{C} E_{vku} \,\middle|\, x(0) = v\right] \tag{23}$$

$$= \sum_{k=1}^{C} \mathbb{E}\left[E_{vku} \mid x(0) = v\right] = \sum_{k=1}^{C} \left(1 - \frac{k-1}{C}\right)\left(\mathcal{T}^k\right)_{uv}. \tag{24}$$

Suppose we run DeepWalk, starting m random walks from each node v, then the expected number of times that u is present in the context of v is given by:

$$\mathbb{E}\left[D_{uv}^{\text{DeepWalk}}\right] = m\mathbb{E}\left[E_{v*u} \mid x(0) = v\right] = m \sum_{k=1}^{C} \left(1 - \frac{k-1}{C}\right)\left(\mathcal{T}^k\right)_{uv}.$$

Finally, we can write down the expectation over the square matrix $\mathbf{D}$:

$$\begin{aligned} \mathbb{E}\left[\mathbf{D}^{\text{DeepWalk}}\right] &= \operatorname{diag}(m, m, \dots, m) \sum_{k=1}^{C} \left(1 - \frac{k-1}{C}\right)\left(\mathcal{T}^k\right) \\ &= \text{Equation (7)} \end{aligned}$$

□