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.

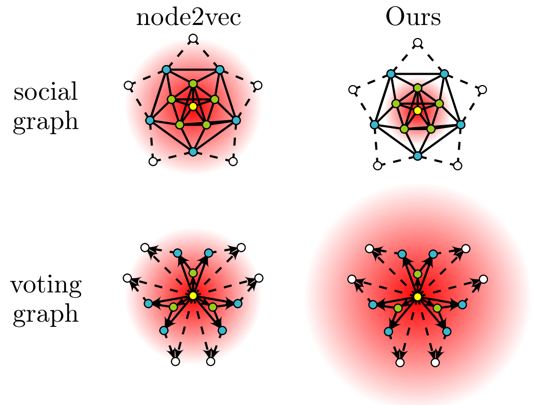


Figure 1: Depiction of how our model assigns context distributions (shaded red) compared to earlier work. We depict the graph from the perspective of anchor node (yellow). Given a social graph (top), where friends of friends are usually friends, our algorithm learns a left-skewed distribution. Given a voting graph (bottom), with general transitivity: $a \rightarrow b \rightarrow c \implies a \rightarrow c$, it learns a long-tail distribution. Earlier methods (e.g. node2vec) use word2vec, which internally uses a linear-decay context distribution, treating all graphs the same.

Introduction

Graph embedding methods based on random walks have demonstrated outstanding performance on a number of tasks including node classification (Perozzi, Al-Rfou, and Skiena, 2014; Grover and Leskovec, 2016), knowledge-graph embedding (Luo et al., 2015), semi-supervised learning (e.g. Yang, Cohen, and Salakhutdinov, 2016), and link prediction (Abu-El-Haija, Perozzi, and Al-Rfou, 2017). These methods operate by simulating many random walks on the graph and produce node embeddings based on co-occurrence statistics. However, 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, Al-Rfou, and Skiena (2014) 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, Goldberg, and Dagan, 2015, 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 (Grover and Leskovec, 2016) 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 and Leskovec (2016) show that different graphs prefer different hyper-parameter choices.

Implicitly, the choice of C, the context distribution (e.g. $\mathcal{U}$), and random-walk hyper-parameters (e.g. node2vec's), all impose a distribution on every node's

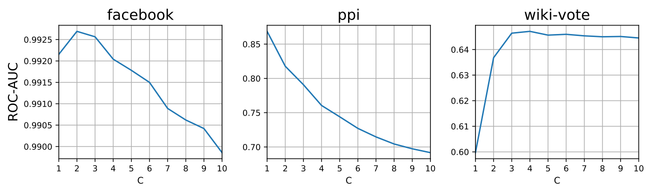


Figure 2: Motivation - Embedding methods are sensitive to hyper-parameters. Every graph prefers its own optimal C value. We plot test ROC-AUC as a function of C using node2vec (Grover and Leskovec, 2016) when embeddings are 128-dimensional. Each point is the average of 7 runs.

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 is replaced with a closed-form expectation over the graph transition matrix. We are able to replace context hyper-parameters by adding attention parameters on coefficients of the power series of the transition matrix. These 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. We show mathematical equivalence between the context distribution hyper-parameters and our attention parameters. 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. Bahdanau, Cho, and Bengio, 2015; Yang et al., 2016), image recognition (Mnih et al., 2014), and detecting rare events in videos (Ramanathan et al., 2016). 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 the state-of-the-art in representation learning for graphs, reducing error by 25%-67%.

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. This includes recent graph convolutional networks (e.g. Niepert, Ahmed, and Kutzkov, 2016; Bruna et al., 2013; Atwood and Towsley, 2016) with spectral variants (Henaff, Bruna, and LeCun, 2015; Defferrard, Bresson, and Vandergheynst, 2016; Bruna et al., 2013), diffusion methods (e.g. Dai, Dai, and Song, 2016; Duvenaud et al., 2015; Chen et al., 2015), including ones trained until fixed-point convergence (Scarselli et al., 2009; Li et al., 2016) and semi-supervised structured learning (Yang, Cohen, and Salakhutdinov, 2016) including spectral method (Kipf and Welling, 2017). We differ from all these methods as we explicitly model the relationship between all node pairs. Our scoring function scores positive edges above negative ones.

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 method directly use the adjacency matrix (Cao, Lu, and Xu, 2016; Wang, Cui, and Zhu, 2016), and others simulate random walks (Perozzi, Al-Rfou, and Skiena, 2014; Grover and Leskovec, 2016; Abu-El-Haija, Perozzi, and Al-Rfou, 2017). Our work falls under the second class of algorithms, where inference is a scoring function $V \times V \rightarrow \mathbb{R}$, trained to score positive edges higher than negative ones. We train our attention random walk while learning embeddings, with an objective that preserves the graph structure.

Preliminaries

Graph Embeddings

We describe Graph-Preserving embedding methods in a general framework (Equation 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) (Koren, Bell, and Volinsky, 2009) 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:

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

Embeddings learned via Random Walks

Introduced by Perozzi, Al-Rfou, and Skiena (2014), this family of methods (incl. Grover and Leskovec, 2016; J et al., 2016) induce random walks along E by starting from a random node $v_0 \in \mathit{sample}(V)$, and repeatedly sampling an edge to transition to next node as $v_{i+1} := \mathit{sample}(E[v_i])$, 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]. where in practice, the context window size c can be sampled from uniform distribution $\mathcal{U}(1, C)$.

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 (Grover and Leskovec, 2016), 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 (Mikolov et al., 2013; Grover and Leskovec, 2016).

Graph Likelihood Abu-El-Haija, Perozzi, and Al-Rfou (2017) learn embeddings by maximizing the **graph likelihood**:

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

where function $g : V \times V \to \mathbb{R}$ 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:

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

where logistic $\sigma(x)$ and indicator $\mathbb{1}[.]$ functions applied element-wise; $\circ$ is an element-wise product; and the $\sum[\mathbf{M}]$ operator computes the sum of all entries in its argument matrix $\mathbf{M}$.

Method

We follow our general framework (see Equation 1), by setting $g(\mathbf{Y}) = g([\mathbf{L} \mid \mathbf{R}]) = \mathbf{L} \times \mathbf{R}^T$ similar to (Koren, Bell, and Volinsky, 2009) and $f(\mathbf{A}) = \mathbb{E}[\mathbf{D}]$, a closed-form expression of the 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 (Equation 5).

Expectation on the Random Walk Matrix

Let $\mathcal{T}$ be the transition matrix for a graph, which can be computed 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 the DeepWalk (Perozzi, Al-Rfou, and Skiena, 2014) 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

[1] Our definition of context differs Perozzi, Al-Rfou, and Skiena (2014) and Grover and Leskovec (2016). 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 the *right stochastic* definition in `https://en.wikipedia.org/wiki/Stochastic_matrix`

$\tilde{\mathbf{P}}^{(0)} \in \mathbb{R}^{|V|\times|V|}$ is the initial starting positions matrix, described in next subsection.

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

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

Here, 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 GloVe (Pennington, Socher, and Manning, 2014), which uses the harmonic series. Using our notation, we can write the expectation on $\mathbf{D}$, obtained by GloVe's decay, as:

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

Choice of $\tilde{\mathbf{P}}^{(0)}$ Random walk simulation of node2vec Grover (2016) starts 80 walks from every graph node $v \in V$. Therefore, in our experiments, we set $\tilde{\mathbf{P}}^{(0)} := \mathrm{diag}(80, 80, \ldots, 80)$. 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 (Grover and Leskovec, 2016; Perozzi, Al-Rfou, and Skiena, 2014). We leave the search for a better $\tilde{\mathbf{P}}^{(0)}$ as future work.

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:

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

Existing random walk models 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, $Q_k = \frac{1}{k}$.

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.

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

$$(Q_1, Q_2, Q_3, \ldots) = \mathrm{softmax}((q_1, q_2, q_3, \ldots)), \quad (10)$$

where the variables q_k are trained via backpropagation. Our hypothesis is as follows. If we don't impose a specific formula on $Q = (Q_1, Q_2, \ldots Q_C)$, 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 $\mathbf{D}^{\mathrm{softmax}[\infty]}$ as[3]:

$$\begin{aligned}\mathbb{E}&\left[\mathbf{D}^{\mathrm{softmax}[\infty]} \;\middle|\; q_1, q_2, q_3, \ldots\right] \\ &= \tilde{\mathbf{P}}^{(0)} \lim_{C\to\infty} \sum_{j=1}^{C} \frac{1}{e^{q_j}} \sum_{k=1}^{C} e^{q_k} (\mathcal{T})^k ,\end{aligned} \quad (11)$$

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

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:

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

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

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

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

[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.

Training Objective

The final training objective for the Softmax attention mechanism, coming from the NLGL equation 5),

$$\min_{\substack{\mathbf{L},\mathbf{R},\\ q_1,q_2,\dots}} \beta||\mathbf{q}||_2^2 - \sum [\mathbf{D}^{\text{softmax}[\infty]} \circ \log\left(\sigma(\mathbf{L}\times\mathbf{R}^T)\right) + \mathbb{1}[\mathbf{A}=0] \circ \log\left(1-\sigma(\mathbf{L}\times\mathbf{R}^T)\right)]$$

is minimized w.r.t node embeddings $\mathbf{L},\mathbf{R} \in \mathbb{R}^{|V|\times\frac{d}{2}}$ and attention parameter vector $\mathbf{q} = (q_1, q_2, \dots)$. $\beta \in \mathbb{R}$ applies L2 regularization on the attention parameters. We emphasize that our attention parameters live within the data $\mathbf{D}$, and are not part of the model $(\mathbf{L},\mathbf{R})$. The constraint $\sum_k Q_k = 1$, through the softmax activation, prevents $\mathbf{D}^{\text{softmax}}$ 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.

Experiments

Link Prediction Experiments

We evaluate the quality of embeddings produced when random walks are augmented with attention, through experiments on link prediction (Liben-Nowell and Kleinberg, 2007). 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 (Perozzi, Al-Rfou, and Skiena, 2014; Grover and Leskovec, 2016). 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 Grover and Leskovec (2016): 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 (Abu-El-Haija, 2017). 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 $g(\mathbf{Y})$ corresponding to positive and negative test edges, then use them to compute ROC AUC. We compare against following baselines:

– **EigenMaps** (Belkin and Niyogi, 2003). 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. $g(\mathbf{Y}) = g([\mathbf{L} \mid \mathbf{R}]) = \mathbf{L}\times\mathbf{R}^T$.

– **DNGR** (Cao, Lu, and Xu, 2016). 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** (Grover and Leskovec, 2016). 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** (Abu-El-Haija, Perozzi, and Al-Rfou, 2017). 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 $Q_3, Q_4, \dots \approx 0$. Figure 4 shows how varying the regularization term β allows the softmax attention model to "attend to" only what each dataset

[5]`http://sami.haija.org/graph/attention.html`

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 (Stark et al., 2006)	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	$\mathbf{93.5 \pm 0.62}$	$\mathbf{93.8 \pm 0.13}$	25.2%
	128	62.2	80.8	55.4	63.7	64.6	91.7	92.9 ± 0.73	$\mathbf{93.8 \pm 0.05}$	25.2%
ego-Facebook	64	96.4	96.7	98.1	99.1	99.0	97.4	$\mathbf{99.3 \pm 0.02}$	$\mathbf{99.4 \pm 0.10}$	33.3%
	128	95.4	94.5	98.4	99.3	99.2	97.3	99.3 ± 0.03	$\mathbf{99.5 \pm 0.03}$	28.6%
ca-AstroPh	64	82.4	91.1	93.9	97.4	96.9	95.7	$\mathbf{98.6 \pm 0.03}$	97.9 ± 0.21	46.2%
	128	82.9	92.4	96.8	97.7	97.5	95.7	$\mathbf{98.6 \pm 0.03}$	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	$\mathbf{93.6 \pm 0.06}$	22.0%
	128	81.2	78.0	89.7	90.1	92.0	90.3	92.2 ± 0.18	$\mathbf{93.9 \pm 0.05}$	23.8%
PPI	64	70.7	75.4	76.7	79.7	70.6	82.4	$\mathbf{90.0 \pm 0.03}$	$\mathbf{89.8 \pm 1.05}$	43.5%
	128	73.7	71.2	76.9	81.8	74.4	83.9	90.4 ± 0.06	$\mathbf{91.0 \pm 0.28}$	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 is calculated as $\frac{(1-\text{us})-(1-\text{them})}{(1-\text{them})}$, where "them" the best performer from prior methods and "us" is the best performer of the attention models.

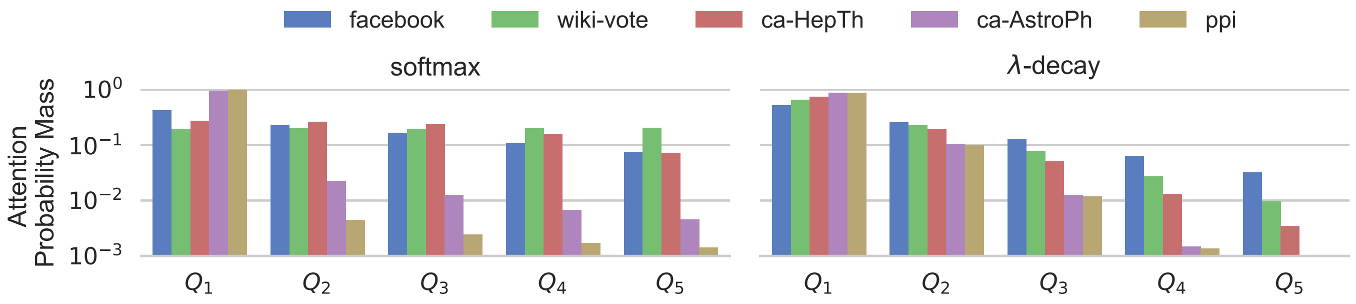


Figure 3: Learned Attention weights Q across datasets for both softmax (left) and λ-decay (right). Auotmatically-found attention agrees with manual tuning node2vec on hyper-parameter C, as shown in Figure 2.

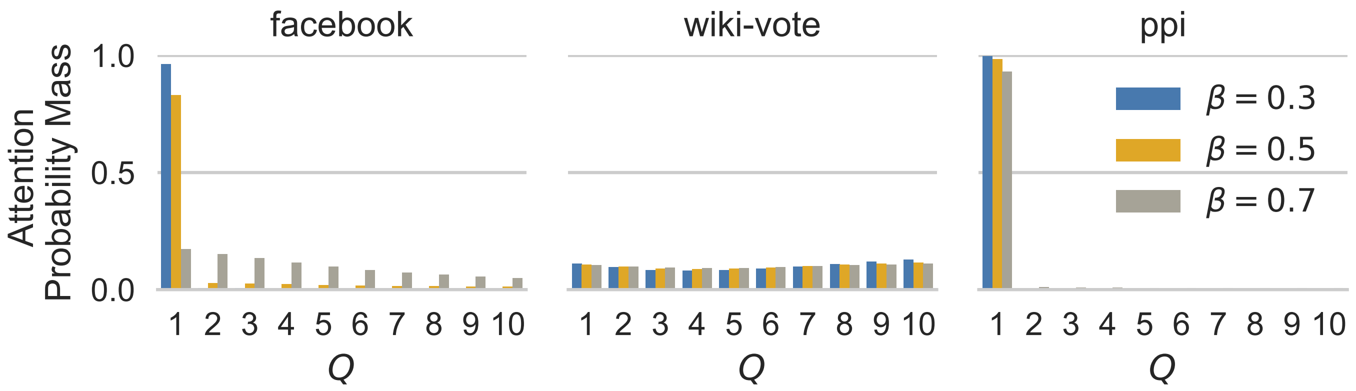
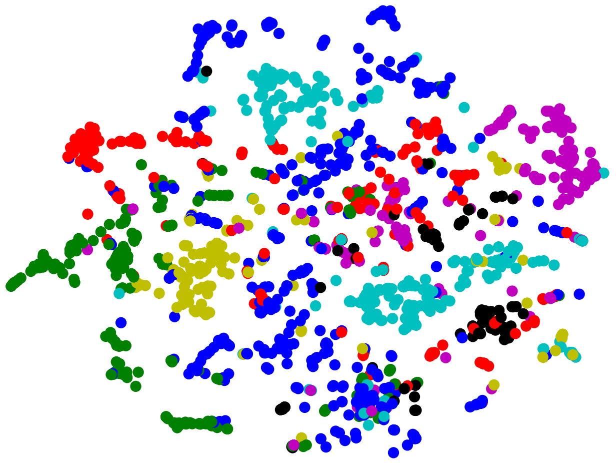


Figure 4: Attention distribution of the softmax model, when varying the regularization (β). Note that distributions can quickly tail off to zero, but some graphs prefer to use the whole space (wiki-vote).


(a) node2vec

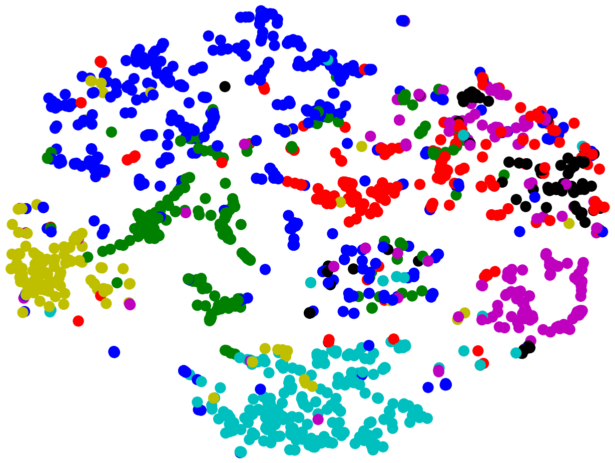

(b) Ours

Figure 5: t-SNE visualization of Cora dataset, trained unsupervised without labels. The labels are used to color nodes.

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.

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 edges (no labels) using our method. Figure 5 shows t-SNE visualization of the Cora dataset, comparing our method with node2vec (Grover and Leskovec, 2016). For classification, we follow the data splits of Yang, Cohen, and Salakhutdinov (2016), 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:

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

where $L_{\text{train}} \in \mathbb{R}^{|V| \times K}$ contains one-hot rows corresponding to nodes in training set, and zero rows corresponding to nodes not in training set. The scalar $\alpha \in \mathbf{R}$ 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 be used for supervised tasks.

Our classification prediciton function, in Equation 14, contains only one scalar parameter α. Its behaves like a "smooth k-nearest-neighbors", as it takes a weighted average of known labels, where the weights are exponential of the dot-product similarity. One should be able to do better by training a neural network over the embedding $\mathbf{Y}$ or the kernel $g(\mathbf{Y})$, and also utilizing node features, as they are available for Cora and Citeseer.

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 Kipf and Welling (2017). We generated results for node2vec and ours, using Equation 14

Nonetheless, we leave semi-supervised learning as future direction, as one possibility would be to integrate our work with the recent Graph Convolutional Networks for Semi-supervised learning (Kipf and Welling, 2017).

Conclusion

We propose to replace hyper-parameters of random walk and context distribution, with an attention model on the random walk, enabling us to learn graph representations which automatically capture intrinsic graph properties. Instead of performing a manual grid search, the distribution of importance for a node's context can now be learned. Our experimental evaluation shows that embeddings learned in this way substantially outperform challenging baselines on a link prediction task, reducing the task error by 20% to 45%. Our visualizations show how the converged attention models convey information about the graph, showing that some graphs (e.g. voting graphs) contain more information on higher powers of the transition matrix $\mathcal{T}$ and better embeddings can be produced when paying more attention to longer walks, while other graphs (e.g. protein graphs) contain more information in short dependencies and require shorter walks.

References

Abu-El-Haija, S.; Perozzi, B.; and Al-Rfou, R. 2017. Learning edge representations via low-rank asymmetric projections. In *ACM International Conference on Information and Knowledge Management (CIKM)*.

Abu-El-Haija, S. 2017. Proportionate gradient updates with percentdelta. In *arXiv*.

Atwood, J., and Towsley, D. 2016. Diffusion-convolutional neural networks. In *Advances in Neural Information Processing Systems (NIPS)*.

Bahdanau, D.; Cho, K.; and Bengio, Y. 2015. Neural machine translation by jointly learning to align and translate. In *International Conference on Learning Representations (ICLR)*.

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

Bruna, J.; Zaremba, W.; Szlam, A.; and LeCun, Y. 2013. Spectral networks and deep locally connected networks on graphs. In *International Conference on Learning Representations*.

Cao, S.; Lu, W.; and Xu, Q. 2016. Deep neural networks for learning graph representations. In *Proceedings of the Association for the Advancement of Artificial Intelligence*.

Chen, L.-C.; Schwing, A.; Yuille, A.; and Urtasun, R. 2015. Learning deep structured models. In *International Conference on Machine Learning*.

Dai, H.; Dai, B.; and Song, L. 2016. Discriminative embeddings of latent variable models for structured data. In *International Conference on Machine Learning*.

Defferrard, M.; Bresson, X.; and Vandergheynst, P. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems (NIPS)*.

Duvenaud, D.; Maclaurin, D.; Iparraguirre, J.; Bombarell, R.; Hirzel, T.; Aspuru-Guzik, A.; and Adams, R. 2015. Convolutional networks on graphs for learning molecular fingerprints. In *Advances in Neural Information Processing Systems (NIPS)*.

Grover, A., and Leskovec, J. 2016. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

Grover, A. 2016. Github: node2vec code. Technical report.

Henaff, M.; Bruna, J.; and LeCun, Y. 2015. Deep convolutional networks on graph-structured data. In *arXiv:1506.05163*.

J, G.; Ganguly, S.; Gupta, M.; Varma, V.; and Pudi, V. 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.

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

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

Levy, O.; Goldberg, Y.; and Dagan, I. 2015. Improving distributional similarity with lessons learned from word embeddings. In *TACL*.

Li, Y.; Tarlow, D.; Brockschmidt, M.; and Zemel, R. 2016. Gated graph sequence neural networks. In *International Conference on Learning Representations*.

Liben-Nowell, D., and Kleinberg, J. 2007. The link-prediction problem for social networks. In *Journal of American Society for Information Science and Technology*.

Luo, Y.; Wang, Q.; Wang, B.; and Guo, L. 2015. Context-dependent knowledge graph embedding. In *Conference on Emperical Methods in Natural Language Processing (EMNLP)*.

Mikolov, T.; Sutskever, I.; Chen, K.; Corrado, G.; and Dean, J. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems NIPS*.

Mnih, V.; Heess, N.; Graves, A.; and kavukcuoglu, k. 2014. Recurrent models of visual attention. In *Advances in Neural Information Processing Systems (NIPS)*.

Niepert, M.; Ahmed, M.; and Kutzkov, K. 2016. Learning convolutional neural networks for graphs. In *International Conference on Machine Learning (ICML)*.

Pennington, J.; Socher, R.; and Manning, C. D. 2014. Glove: Global vectors for word representation. In *Conference on Empirical Methods in Natural Language Processing, EMNLP*.

Perozzi, B.; Al-Rfou, R.; and Skiena, S. 2014. Deepwalk: Online learning of social representations. In *Knowledge Discovery and Data Mining (KDD)*.

Ramanathan, V.; Huang, J.; Abu-El-Haija, S.; Gorban, A.; Murphy, K.; and Fei-Fei, L. 2016. Detecting events and key actors in multi-person videos. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Scarselli, F.; Gori, M.; Tsoi, A.; Hagenbuchner, M.; and Monfardini, G. 2009. The graph neural network model. In *IEEE Trans. on Neural Networks*.

Stark, C.; Breitkreutz, B.; Reguly, T.; Boucher, L.; Breitkreutz, A.; and Tyers, M. 2006. Biogrid: A general repository for interaction datasets. In *Nucleic Acids Research*.

Wang, D.; Cui, P.; and Zhu, W. 2016. Structural deep network embedding. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

Yang, Z.; Yang1, D.; Dyer, C.; He, X.; Smola, A.; and Hovy, E. 2016. Hierarchical attention networks for document classification. In *Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*.

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