Dynamic Embedding on Textual Networks via a Gaussian Process

Pengyu Cheng, Yitong Li, Xinyuan Zhang, Liqun Chen, David Carlson, Lawrence Carin
Duke University
pengyu.cheng@duke.edu

Abstract

Textual network embedding aims to learn low-dimensional representations of text-annotated nodes in a graph. Prior work in this area has typically focused on fixed graph structures; however, real-world networks are often dynamic. We address this challenge with a novel end-to-end node-embedding model, called Dynamic Embedding for Textual Networks with a Gaussian Process (DetGP). After training, DetGP can be applied efficiently to dynamic graphs without re-training or backpropagation. The learned representation of each node is a combination of *textual* and *structural* embeddings. Because the structure is allowed to be dynamic, our method uses the Gaussian process to take advantage of its non-parametric properties. To use both local and global graph structures, diffusion is used to model multiple hops between neighbors. The relative importance of global versus local structure for the embeddings is learned automatically. With the non-parametric nature of the Gaussian process, updating the embeddings for a changed graph structure requires only a forward pass through the learned model. Considering link prediction and node classification, experiments demonstrate the empirical effectiveness of our method compared to baseline approaches. We further show that DetGP can be straightforwardly and efficiently applied to dynamic textual networks.

1 Introduction

Learning latent representations for graph nodes has attracted considerable attention in machine learning, with applications in social networks (Fan et al. 2019; Perozzi, Al-Rfou, and Skiena 2014), knowledge bases (Trivedi et al. 2017), recommendation systems (Ying et al. 2018), and bioinformatics (Zitnik and Leskovec 2017). *Textual networks* additionally contain rich semantic information, so text can be included with the graph structure to predict downstream tasks, such as link prediction (Zhang et al. 2018a), node classification (Kipf and Welling 2017), and graph generation (Kipf and Welling 2016). For instance, social networks have links between users, and typically each user has a profile (text). The goal of *textual network embedding* is to learn node embeddings by jointly considering *textual* and *structural* information in the graph.

Most of the aforementioned textual network embedding methods focus on a fixed graph structure (Tu et al. 2017; Zhang et al. 2018a; Shen et al. 2019). When new network nodes are added to the graph, these frameworks require that the whole model be re-trained to update the existing nodes and add representations for the new nodes, leading to high computational complexity. However, networks are often dynamic; in social networks, users and relationships between users change over time ($e.g.$, new users, new friends, unfriending, $etc.$). It is impractical to update the full model whenever a new user is added. This paper seeks to address this challenge and learn an embedding method that adapts to a changed graph, without re-training.

Prior dynamic embedding methods usually focus on predicting how the graph structure changes over time, by training on multiple time steps (Zhou et al. 2018; Seo et al. 2018; Du et al. 2018). In such a model, a dynamic network embedding is approximated by multiple steps of fixed network embeddings. In contrast, our method only needs to train on a single graph and it can quickly adapt to related graph structures. Additionally, in prior work textual information is rarely included in dynamic graphs. Two exceptions have looked at dynamic network embeddings with changing node attributes (Li et al. 2017; 2018). However, both require pre-trained node features, whereas we show that it is more powerful to learn the text encoder in a joint framework with the structural embedding.

We propose **D**ynamic **E**mbedding for **T**extual Networks with a **G**aussian **P**rocess (DetGP), a novel end-to-end model for performing unsupervised dynamic textual network embedding. DetGP learns *textual* and *structural* features jointly for both fixed and dynamic networks. The textual features indicate the intrinsic attributes of each node based on the text alone, while the structural features reveal the node relationships of the whole community. The structural features utilize both the textual features and the graph topology. This is achieved by smoothing the kernel function in a Gaussian process (GP) with a multi-hop graph transition matrix. This GP-based structure can handle newly added nodes or dynamic edges due to its non-parametric properties (Rasmussen 2006). To facilitate fast computation, we learn inducing points to serve as landmarks in the GP (Yu and Chu

2008). Since the inducing points are fixed after training, computing new embeddings only requires calculating similarity to the inducing points, alleviating computational issues caused by changing graph structure.

To evaluate the proposed approach, the learned node embeddings are used for link prediction and node classification. Performance on those downstream tasks demonstrates that learned embeddings capture relevant information. We also perform these tasks on dynamic textual networks and visualize the learned inducing points. Empirically, DetGP outperforms other models in downstream tasks, yielding efficient and accurate predictions on newly added nodes.

2 Related Work

Textual Network Embedding: Many graphs have rich text information for each node (Yang et al. 2015) (*e.g.*, paper abstracts in citation networks, user profiles in social networks, product and costumer descriptions in online shopping, *etc.*), which leads to the problem of *textual network embedding* (Tu et al. 2017). Textual network embedding learns enhanced node representations by combining the textual embeddings and structural embeddings together. Textual embeddings are learnt by encoding raw-text via text encoders. Structural are obtained based on the connection information on graphs. Recent textual embedding methods (Zhang et al. 2018a; Shen et al. 2019; Tu et al. 2017; Sun et al. 2016) learn both textual and structural embeddings jointly in an end-to-end training process with neural networks. However, previous methods require all the network connection information before training, which makes them difficult to apply in dynamic network scenarios.

Dynamic Network Embedding: Graph structures are often *dynamic* (*e.g.*, paper citation increasing or social relationship changing overtime), but fixed network embedding algorithms require re-training when graphs change. This issue is addressed by dynamic network embeddings (Trivedi et al. 2018). Research on dynamic graphs has usually focused only on *structural embeddings* (Trivedi et al. 2018; Du et al. 2018; Guo, Xu, and Chen 2018) without considering rich side information associate with vertices. While these methods learn how to update the model as the graph changes, they are not real-time algorithms, instead using gradient back-propagation to update representations when new nodes are added or the graph structure changes (Goyal et al. 2018; Du et al. 2018).

Gaussian Process: A Gaussian Process (GP) $f(\boldsymbol{x})$ is a collection of random variables such that any finite subset of those variables are Gaussian distributed. More specifically, given finite set $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n\}$ with $n \in \mathbb{N}_+$, the corresponding signal $[f(\boldsymbol{x}_1), f(\boldsymbol{x}_2), \ldots, f(\boldsymbol{x}_n)]^\intercal \sim \mathcal{N}([m(\boldsymbol{x}_1), m(\boldsymbol{x}_2), \ldots, m(\boldsymbol{x}_n)]^\intercal, [k(\boldsymbol{x}_i, \boldsymbol{x}_j)]_{n\times n})$, where $m(\boldsymbol{x})$ is a mean function and $k(\cdot,\cdot)$ is a covariance kernel function. GPs are used widely in the Bayesian machine learning literature as priors on functions (Titsias 2009; Rasmussen 2006; Titsias and Lawrence 2010). To learn the mapping $y = f(\boldsymbol{x})$ with training data $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^N$ and unlabeled testing data $\{\boldsymbol{x}'_j\}_{j=1}^M$, with Gaussian process as the prior, $[f(\boldsymbol{x}_1), f(\boldsymbol{x}_2) \ldots, f(\boldsymbol{x}_n), f(\boldsymbol{x}'_1), \ldots, f(\boldsymbol{x}'_M)]$ follows a multivariate Gaussian distribution. Given observations $f(\boldsymbol{x}_i) = y_i$, the conditional distribution for $[f(\boldsymbol{x}'_1), \ldots, f(\boldsymbol{x}'_M)]$ can be easily obtained, which is also Gaussian distributed. Recently, GPs have been used for node embedding learning on graphs (Ma, Cui, and Zhu 2018; Ng, Colombo, and Silva 2018). However, these works require pre-trained node features, leading to a two-step training process. In contrast, our proposed model is an end-to-end framework that can jointly train the feature extractor (text encoder) with the GP parameters.

3 Model

We assume the input data are given as an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{v_n\}_{n=1}^N$ is the node set and $\mathcal{E} = \{(v_n, v_{n'}) : v_n, v_{n'} \in \mathcal{V}\}$ is the edge set. Each node v_n has an associated L_n-length text sequence $\boldsymbol{t}_n = \{w_i\}_{i=1}^{L_n}$, where each w_i is a natural language word. The adjacency matrix $\boldsymbol{A} \in \{0,1\}^{N\times N}$ represents node relationships, where $A_{nn'} = 1$ if $(v_n, v_{n'}) \in \mathcal{E}$ and $A_{nn'} = 0$ otherwise. Our objective is to learn a low-dimensional embedding vector $\boldsymbol{h}_n$ for each node $v_n \in \mathcal{V}$ that captures both textual and structural features of the graph $\mathcal{G}$.

Figure 1 gives the framework of the proposed model, DetGP. Text $\boldsymbol{t}_n$ is input to a text encoder $\boldsymbol{g}_{\boldsymbol{\theta}}(\cdot)$ with parameters $\boldsymbol{\theta}$, described in Section 3.1. The output $\boldsymbol{x}_n = \boldsymbol{g}_{\boldsymbol{\theta}}(\boldsymbol{t}_n)$ is the *textual embedding* of node v_n. This textual embedding is both part of the complete embedding $\boldsymbol{h}_n$ and an input into the structural embedding layer (dotted purple box) that is combined with the graph structure in a GP framework, discussed in Section 3.2. In addition, multiple hops are modeled in this embedding layer to better reflect the graph architecture and use both local and global graph structure. The mathematical analysis of the structural embeddings is given in Section 3.3. To scale up the model to large datasets, we adopt the idea of inducing points (Titsias 2009; Titsias and Lawrence 2010), which serve as grid points in the model. The output structural embeddings are denoted as $\boldsymbol{s}_n$, which are combined to form the complete node embedding $\boldsymbol{h}_n = [\boldsymbol{x}_n; \boldsymbol{s}_n]$.

The model is trained by using the negative sampling loss (Tu et al. 2017), where neighbor nodes should be more similar than non-neighbor nodes. The learning procedure is fully described in Section 3.4. When the graph structure is updated, *i.e.* new nodes v_{new} with text $\boldsymbol{t}_{new}$ are added or links are changed, the node embeddings are updated by a single forward-propagation step without relearning any model parameters. This property comes from the non-parametric nature of the GP-based structure, and it greatly increases efficiency for dynamic graphs.

3.1 Text Encoder

There are many existing text encoders (Melamud, Goldberger, and Dagan 2016; Cer et al. 2018; Lin et al. 2017), often based on deep neural networks. However, using a deep neural network encoder can overfit on graphs because of the relatively small size of textual data (Zhang et al. 2018a). Therefore, various encoders are proposed to extract rich textual information specifically from graphs (Tu et al. 2017;

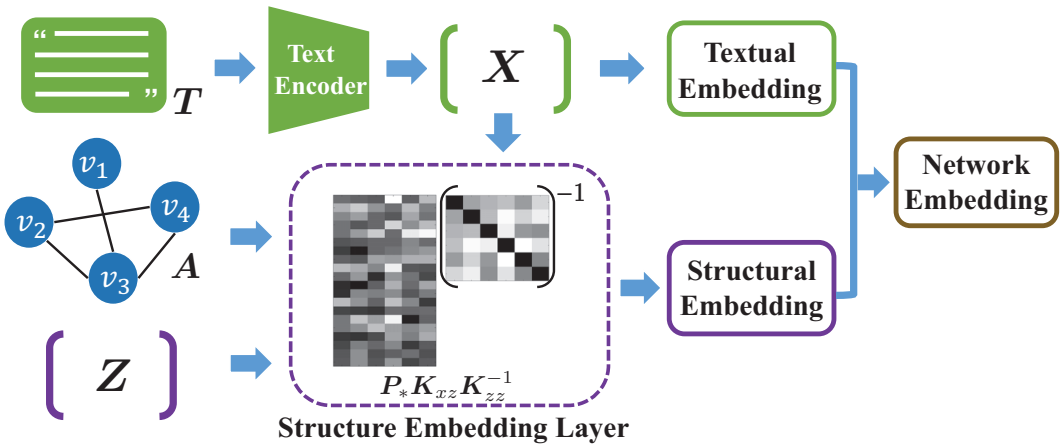

Figure 1: The input of the DetGP is the connection information $\boldsymbol{A}$ of the network and textual side information $\boldsymbol{T} = \{\boldsymbol{t}_n\}_{n=1}^N$ for nodes. The model first encodes text $\boldsymbol{T}$ to a low-dimension representation $\boldsymbol{X} = \{\boldsymbol{x}_n\}_{n=1}^N$, then infers the structural embeddings by $\boldsymbol{X}$ and $\boldsymbol{A}$ via a Gaussian process. Inducing points $\boldsymbol{Z} = \{\boldsymbol{z}_m\}_{m=1}^M$ are used to reduce computational complexity. The output network embeddings combine the textual and structural embeddings.

Zhang et al. 2018a; Shen et al. 2019). In general, we aim to learn a text encoder $\boldsymbol{g_\theta}$ with parameters $\boldsymbol{\theta}$ that encodes semantic features $\boldsymbol{x}_n = \boldsymbol{g_\theta}(\boldsymbol{t}_n)$. A simple and effective text encoder is the word embedding average (Wavg) $\boldsymbol{g_\theta}(\boldsymbol{t}) = \frac{1}{L_n}\sum_{i=1}^{L_n} \boldsymbol{\nu}_i$, where $\boldsymbol{\nu}_i$ is the corresponding embedding of w_i from the sequence $\boldsymbol{t}_n = \{w_i\}_{i=1}^{L_n}$. This is implemented by a learnable look-up table. (Zhang et al. 2018a) proposed a diffused word average encoder (DWavg) to leverage textual information over multiple hops on the network. Because DetGP focuses mainly on the structural embeddings, we do not focus on developing a new text encoder. Instead, we show that DetGP has compatibility with different text encoders, and our experiments use these two text encoders (Wavg and DWavg).

3.2 Structural Embedding Layer

The structural embedding layer transforms the encoded text feature $\boldsymbol{x}$ to structural embedding $\boldsymbol{s}$ using a GP in conjunction with the graph topology. Before introducing the GP, we introduce the multi-hop transition matrix $\boldsymbol{P}_*$ that will smooth the GP kernel.

Multi-hop Transition Matrix: In a graph, one node can be directly or indirectly connected to others by edges. Only considering direct connections is limiting. For example, in citation networks, cited papers should be closely related to a manuscript, so considering both a neighbor and its neighbors should add information. Following this intuition, we propose to use multiple hops, or multi-step graph random walks, to model both local and global structure.

Suppose $\boldsymbol{P}$ is the normalized transition matrix, $i.e.$ a normalized version of $\boldsymbol{A}$ where each row sums to one and $P_{nn'}$ represents the probability of a transition from node n to node n'. If $\boldsymbol{P}$ represents the transition from a single hop, then higher orders of $\boldsymbol{P}$ will give multi-hop transition probabilities. Specifically, $\boldsymbol{P}^j$ is the jth power of $\boldsymbol{P}$, where $P^j_{nn'}$ gives the probability of transitioning from node n to node n' after j random hops on the graph (Abu-El-Haija et al. 2018). Different powers of $\boldsymbol{P}$ provide different levels of smoothing on the graph, and vary from using local to global structure. *A priori* though, it is not clear what level of structure is most important for learning the embedding. Therefore, they are combined in a learnable weighting scheme. Denoting the weights as $\boldsymbol{\alpha}$, this is

$$\boldsymbol{P}_* = \textstyle\sum_{j=0}^{J} \alpha_j \boldsymbol{P}^j, \quad s.t. \sum_{j=0}^{J} \alpha_j = 1, \qquad (1)$$

where J is the maximum number of steps considered. The constraint that $\sum_{j=0}^{J} \alpha_j = 1$ in (1) is implemented by a softmax function. Note that $\boldsymbol{P}^0 = \boldsymbol{I}_N$ is an identity matrix, that treats each node independently. In contrast, a large power of $\boldsymbol{P}$ would typically be very smooth after taking many hops. Therefore, $\boldsymbol{\alpha}$ can learn the importance of local ($\boldsymbol{P}^0$ or $\boldsymbol{P}^1$) and global (large powers of $\boldsymbol{P}$) graph structure for the node embeddings. Equation (1) can be viewed as a generalized form for DeepWalk (Perozzi, Al-Rfou, and Skiena 2014) or Node2vec (Grover and Leskovec 2016). In practice, learning the weights appears to be more robust than hand-engineering them (Abu-El-Haija et al. 2018).

GP Structural Embedding Prior: To describe the other key components of the structural embedding layer, we describe the GP approach. We define a latent function $f(\boldsymbol{x})$ over the textual embedding $\boldsymbol{x}$ with a GP prior $f(\boldsymbol{x}) \sim \mathcal{GP}(\boldsymbol{0}, k(\boldsymbol{x}_n, \boldsymbol{x}_{n'}))$. Inspired by (Ng, Colombo, and Silva 2018), instead of using this GP directly to determine the embedding, the learned graph diffusion is used on top of this Gaussian process. For finite samples, the combination of the graph diffusion and the GP yields a conditional structural embedding that can be expressed as a multivariate Gaussian distribution:

$$p([s_{1i}, \ldots, s_{Ni}]^\intercal | [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N]) = \mathcal{N}(\boldsymbol{0}, \boldsymbol{P}_*^\intercal \boldsymbol{K}_{XX} \boldsymbol{P}_*), \qquad (2)$$

where $[\boldsymbol{K}_{XX}]_{nn'} = k(\boldsymbol{x}_n, \boldsymbol{x}_{n'})$ and i is a index of our structure embedding feature. Each dimension of the structural embedding follows this Gaussian distribution with the

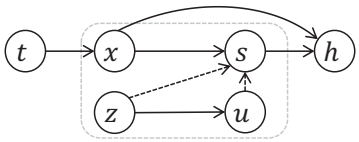

Figure 2: Model flow chart. Inducing points are denoted as $\boldsymbol{z}$, with embedding $\boldsymbol{u}$. The final node embedding $\boldsymbol{h}$ is the concatenation of text embedding $\boldsymbol{x}$ and structural embedding $\boldsymbol{s}$.

same covariance matrix $\boldsymbol{K}_{XX}$ smoothed by $\boldsymbol{P}^*$. In practice, we use the first-degree polynomial kernel

$$k\left(\boldsymbol{x}_n, \boldsymbol{x}_{n'}\right) = \boldsymbol{x}_n^\intercal \boldsymbol{x}_{n'} + C,\ C > 0 \tag{3}$$

because it outperforms others due to its numerical stability. Moreover, the linear kernel in (3) speeds up computation and increases model stability.

Inducing Points: GP models are known to suffer from computational complexity with large data size N. To scale up the model, we use the inducing points based on the Sparse pseudo-inputs Gaussian process (SPGP) (Snelson and Ghahramani 2006). Let $\boldsymbol{Z} = [\boldsymbol{z}_1, \cdots, \boldsymbol{z}_M]^\intercal$ with $M < N$ denote inducing points (pseudo-textual embeddings) in the same space as the textual features. Assume $\boldsymbol{U} = [\boldsymbol{u}_1, \cdots, \boldsymbol{u}_M]^\intercal$ are corresponding the pseudo-structural embeddings of $\boldsymbol{Z}$, which is a function of $\boldsymbol{z}$ following the same GP function. The structural and textual embeddings of real data samples are denoted as $\boldsymbol{S} = [\boldsymbol{s}_1, \cdots, \boldsymbol{s}_N]^\intercal$ and $\boldsymbol{X} = [\boldsymbol{x}_1, \cdots, \boldsymbol{x}_N]^\intercal$. Given the inducing points, the conditional distribution of our structural embeddings is

$$\begin{aligned} &p(\boldsymbol{S}_i|\boldsymbol{X}, \boldsymbol{Z}, \boldsymbol{U}) = \mathcal{N}\left(\boldsymbol{\mu}_{S_i|Z}, \boldsymbol{\Sigma}_{S|Z}\right), \\ &\boldsymbol{\mu}_{S_i|Z} = \boldsymbol{P}_*^\intercal \boldsymbol{K}_{XZ}\left(\boldsymbol{K}_{ZZ} + \sigma \boldsymbol{I}_M\right)^{-1} \boldsymbol{U}_i, \\ &\boldsymbol{\Sigma}_{S|Z} = \boldsymbol{P}_*^\intercal \boldsymbol{K}_{XX} \boldsymbol{P}_* - \boldsymbol{P}_*^\intercal \boldsymbol{K}_{XZ}\left(\boldsymbol{K}_{ZZ} + \sigma \boldsymbol{I}_M\right)^{-1} \boldsymbol{K}_{ZX} \boldsymbol{P}_*, \end{aligned} \tag{4}$$

where $[\boldsymbol{K}_{XZ}]_{nm} = k(\boldsymbol{x}_n, \boldsymbol{z}_m)$ and $[\boldsymbol{K}_{ZZ}]_{mm'} = k(\boldsymbol{z}_m, \boldsymbol{z}_{m'})$. The subscript i indicates the ith column of a matrix ($\boldsymbol{S}_i$ is the concatenation of the ith element from all node structural embeddings). Each dimension of $\boldsymbol{S}$ has a multivariate Gaussian distribution with unique mean value $\boldsymbol{\mu}_{s_i|Z}$ but the same covariance $\boldsymbol{\Sigma}_{S|Z}$. Note that a small number σ is added to the diagonal elements of the kernel $\boldsymbol{K}_{ZZ}$ to enhance model stability. The relationships of different parameters are given in Figure 2. The inducing points $\{\boldsymbol{z}_m\}_{m=1}^M$ and text features $\{\boldsymbol{x}_n\}_{n=1}^N$ share the same textual embedding space; the pseudo-textual embeddings $\{\boldsymbol{u}_m\}_{m=1}^M$ and the structural embeddings $\{\boldsymbol{s}_n\}_{n=1}^n$ share the same structural embedding space.

During the training, the textual embedding space is continuously changed, because the text encoder $\boldsymbol{g_\theta}(\boldsymbol{t})$ is updated by the gradient descent algorithm iterative. Therefore, the inducing points in the textual embedding space should also be updated correspondingly. To obtain the optimal inducing points $\boldsymbol{Z}$ with corresponding pseudo-structural embeddings $\boldsymbol{U}$, we jointly train them using gradient descent with weights in other layers. To back-propagate through the stochastic posterior distribution $p(\boldsymbol{S}_i|\boldsymbol{X}, \boldsymbol{Z}, \boldsymbol{U})$ in equation (4), we propose the mean approximation strategy. We use the mean of $p(\boldsymbol{S}_i|\boldsymbol{X}, \boldsymbol{Z}, \boldsymbol{U})$, $\hat{\boldsymbol{S}} = \boldsymbol{P}_*^\intercal \boldsymbol{K}_{XZ}\left(\boldsymbol{K}_{ZZ} + \sigma \boldsymbol{I}_M\right)^{-1} \boldsymbol{U}$, as unbiased estimation of structural embeddings. Then gradients from $\boldsymbol{S}$ can be easily propagated to $\boldsymbol{Z}$ and $\boldsymbol{U}$.

3.3 Analysis of the Structural Embedding

We analyze the kernel function in (2) and (3) to show how the graph structure is used in the embedding layer. Denote $P_{nn'}^j$ in equation (1) as the transition probability from node n to node n' in j hops, then the correlation between node n and n' in (2) can be expanded as

$$\begin{aligned} \operatorname{cov}(s_{ni}, s_{n'i}) = &\alpha_0^2 k(\boldsymbol{x}_n, \boldsymbol{x}_{n'}) + \alpha_0 \sum_{j=1}^{J} \alpha_j \sum_{r=1}^{N} P_{nr}^j k(\boldsymbol{x}_n, \boldsymbol{x}_r) \\ &+ \alpha_0 \sum_{j=1}^{J} \alpha_j \sum_{r=1}^{N} P_{n'r}^j k(\boldsymbol{x}_{n'}, \boldsymbol{x}_r) \\ &+ \sum_{j=1}^{J} \sum_{j'=1}^{J} \sum_{\substack{r=1 \\ r \neq n,n}}^{N} \sum_{\substack{r'=1 \\ r' \neq n,n'}}^{N} \alpha_j \alpha_{j'} P_{nr}^j P_{n'r'}^{j'} k(\boldsymbol{x}_{n'}, \boldsymbol{x}_{r'}) \end{aligned} \tag{5}$$

The covariance is the same for all indices i. The first term in (5) measures the kernel function between $\boldsymbol{x}_n$ and $\boldsymbol{x}_{n'}$. The next two terms show the relationship between $\boldsymbol{x}_n$ and the weighted multi-hop neighbors of $\boldsymbol{x}_{n'}$ and vice versa. $\boldsymbol{\alpha}$ controls how much different hops are used. The last term is the pairwise-weighted higher order relationship between any two nodes in the graph, except n and n'. The covariance structure uses the whole graph and learns how to balance local and global information. If node n has no edges, then it will not be influenced by other nodes besides textual similarity. In contrast, a node with dense edge connections will be smoothed by its neighbors.

With the inducing points $[\boldsymbol{z}_1, \boldsymbol{z}_2, \ldots, \boldsymbol{z}_M]$, equation (5) can be modified as $\operatorname{cov}(s_{ni}, u_{mi}) = \alpha_0 k\left(\boldsymbol{x}_n, \boldsymbol{z}_m\right) + \sum_{j=1}^{J} \alpha_j \sum_{n'=1}^{N} P_{nn'}^j k\left(\boldsymbol{x}_{n'}, \boldsymbol{z}_m\right)$. The covariance between node v_n and the inducing points includes the local information $k\left(\boldsymbol{x}_n, \boldsymbol{z}_m\right)$, as well as the smoothed effect from $\sum_{j=1}^{J} \alpha_j \sum_{n'=1}^{N} P_{nn'}^j k\left(\boldsymbol{x}_{n'}, \boldsymbol{z}_m\right)$. This can also be viewed as feature smoothing over neighbors. Since inducing points do not contain links to other inducing points, there is no smoothing function for them. Each inducing point can be viewed as a node that already includes global graph information.

3.4 Algorithm Outline

The structural embedding $\boldsymbol{s}_n$ and the textual embedding $\boldsymbol{x}_n$ are concatenated to form the final node embedding $\boldsymbol{h}_n = [\boldsymbol{x}_n; \boldsymbol{s}_n]$. If desired, $\boldsymbol{h}_n$ can then pass through several additional fully connected layers; for simplicity, we consider this the final embedding form. To learn the embeddings in the unsupervised framework, most existing works adopt the technique of negative sampling (Zhang et al. 2018a; 2018b; Tu et al. 2017), that tries to maximize the conditional probability of one node's embedding given its one- and multi-hop neighbors, while maintaining a low conditional probability for non-neighbors. In the proposed framework, this loss is

given for a single hop,

$$\mathcal{L} = -\frac{1}{|\mathcal{E}|}\sum_{(v_n,v_{n'})\in\mathcal{E}}\log\left(\sigma(\boldsymbol{h}_n^\intercal\boldsymbol{h}_{n'})\right) - \frac{1}{N_s}\sum_{(v_n,v_{n'})\notin\mathcal{E}}\log\left[1-\sigma(\boldsymbol{h}_n^\intercal\boldsymbol{h}_{n'})\right]. \quad (6)$$

N_s is a weighting constant and can be set as $N_s = \#\{(v_n, v_{n'}) \notin \mathcal{E}\}$. In practice, using all nodes is infeasible, so a subset of neighbors and non-neighbors will be sampled. Equation (6) maximizes the inner product among neighbors in the graph while minimizing the similarity among non-neighbors. Our model is trained end-to-end by taking gradients of loss $\mathcal{L}$ with respect to $\boldsymbol{\theta}$, $\boldsymbol{Z}$ and $\boldsymbol{U}$. The inducing points are initialized as the k-means centers of the encoded text features. Then, $\boldsymbol{Z}$ and the text encoder are trained jointly to minimize the loss function. Note that our model can also take a mini-batch of nodes as in GraphSage (Hamilton, Ying, and Leskovec 2017). Adam (Kingma and Ba 2015) is used to optimize the parameters.

For a new node v_{new} with text $\boldsymbol{t}_{new}$, the transition matrix $\boldsymbol{P}_{new}$ is first updated, and the embeddings can be applied directly without additional back-propagation. Specifically, we first compute $\boldsymbol{x}_{new} = \boldsymbol{g}_\theta(\boldsymbol{t}_{new})$ from the text encoder. Then with $[\boldsymbol{x}, \boldsymbol{x}_{new}]$, the structural embedding of all nodes can be computed as $[\boldsymbol{s}, \boldsymbol{s}_{new}] = \sum_{j=0}^{J}\alpha_j\boldsymbol{P}_{new}^j\boldsymbol{K}_{x_{new}Z}(\boldsymbol{K}_{ZZ} + \sigma\boldsymbol{I}_M)^{-1}\boldsymbol{U}$. During this process, the structural embeddings of the original nodes also update due to the change in graph structure.

4 Experiments

To demonstrate the effectiveness of our DetGP embeddings, we conduct experiments first with static networks as in (Tu et al. 2017; Zhang et al. 2018a; Shen et al. 2019), and then on dynamic graphs. In the standard setup, learning textual network embeddings requires that all nodes are available in the graph. In contrast, we demonstrate in Section 4.3 that our proposed model can infer embeddings efficiently after training on newly added nodes. We evaluate the graph embeddings on link prediction and node classification on the following real-world datasets:

- **Cora** is a paper citation network, with a total of 2,277 vertices and 5,214 edges in the graph, where only nodes with text are kept. Each node has a text abstract about machine learning and belongs to one of seven categories.
- **DBLP** is a paper citation network with 60,744 nodes and 52,890 edges. Each node represents one paper in computer science in one of four categories: database, data mining, artificial intelligence, and computer vision.
- **HepTh** (High Energy Physics Theory) (Leskovec, Kleinberg, and Faloutsos 2007) is another paper citation network. The original dataset contains 9,877 nodes and 25,998 edges. We only keep nodes with associated text, so this is limited to 1,038 nodes and 1,990 edges.

For a fair comparison with previous work, we follow the setup in (Tu et al. 2017; Zhang et al. 2018a; Shen et al. 2019), where the embedding for each node has dimension 200, a concatenation of a 100-dimensional textual embedding and a 100-dimensional structural embedding. We evaluate our DetGP base on two text encoders: the word embedding average (Wavg) encoder and the diffused word embedding average (DWavg) encoder from Zhang et al. (2018a), introduced in Section 3.1. The maximum number of hops J in $\boldsymbol{P}^*$ is set to 3.

4.1 Link Prediction

The link prediction task seeks to infer if two nodes are connected, based on the learned embeddings. This standard task tests if the embedded node features contain graph connection information. For a given network, we randomly keep a certain percentage (15%, 35%, 55%, 75%, 95%) of edges and learn embeddings. At test time, we calculate the inner product of pairwise node embedding. A large inner product value indicates a potential edge between two nodes. The AUC score (Hanley and McNeil 1982) is computed in this setting to evaluate the performance. The results are shown in Table 1 on Cora and HepTh. Since the DBLP dataset only has 52,890 edges which is far too sparse compared with the node number 60,744, we do not evaluate the AUC score on it as a consequence of high variance from sampling edges. The first four models only embed structural features, while the remaining alternatives use both textual and structural embeddings. We also provide the DetGP results of with only textual embeddings and only structure embeddings for ablation study.

From Table 1, adding textual information in the embedding can improve the link prediction result by a large margin. Even using only textual embeddings, DetGP gains significant improvement compared with only structure-based methods, and achieves competitive performance compared with other text-based embedding methods. Using only structural information is slightly better than using only textual embeddings, since link prediction is a more structure-dependent task, which also indicates that DetGP learns inducing points $\boldsymbol{Z}$ that can effectively represent the network structure. Compared with other textual network embedding methods, DetGP has very competitive AUC scores, especially when only given a small percentage of edges. Noting that for our methods the text encoders come from the baselines Wavg and DWavg (Zhang et al. 2018a), the performance gain should come from the proposed structural embedding framework.

4.2 Node Classification

Node classification tasks aim to predict the category labels of the nodes based on the network structure and the node attributes. For textual networks, node classification requires high-quality *textual embeddings* because *structural embeddings* alone do not accurately reflect node categories. Therefore, we only compare to methods designed for textual network embedding. After training converges, a linear SVM classifier is learned on the trained node embeddings and performance is estimated by a hold-out set. In Table 2, we compare our methods (Wavg+DetGP, DWavg+DetGP) with recent textual network embedding methods under different proportions (10%, 30%, 50%, 70%) of given nodes in the training set. Following the setup in Zhang et al. (2018a), the evaluation metric is Macro-F1 score (Powers 2011). We

	Cora					HepTh				
%Training Edges	**15%**	**35%**	**55%**	**75%**	**95%**	**15%**	**35%**	**55%**	**75%**	**95%**
MMB (Airoldi et al. 2008)	54.7	59.5	64.9	71.1	75.9	54.6	57.3	66.2	73.6	80.3
node2vec (Grover and Leskovec 2016)	55.9	66.1	78.7	85.9	88.2	57.1	69.9	84.3	88.4	89.2
LINE (Tang et al. 2015)	55.0	66.4	77.6	85.6	89.3	53.7	66.5	78.5	87.5	87.6
DeepWalk (Perozzi, Al-Rfou, and Skiena 2014)	56.0	70.2	80.1	85.3	90.3	55.2	70.0	81.3	87.6	88.0
TADW (Yang et al. 2015)	86.6	90.2	90.0	91.0	92.7	87.0	91.8	91.1	93.5	91.7
CANE (Tu et al. 2017)	86.8	92.2	94.6	95.6	97.7	90.0	92.0	94.2	95.4	96.3
DMATE (Zhang et al. 2018a)	91.3	93.7	96.0	97.4	98.8	NA	NA	NA	NA	NA
WANE (Shen et al. 2019)	91.7	94.1	96.2	**97.5**	**99.1**	92.3	95.7	97.5	97.7	**98.7**
DetGP (Wavg) only Text	83.4	89.1	89.9	90.9	92.3	86.5	89.6	90.2	91.5	92.6
DetGP (Wavg) only Struct	85.4	89.7	91.0	92.7	94.1	89.7	92.1	93.5	94.8	95.1
DetGP (Wavg)	92.8	94.8	95.5	96.2	97.5	93.2	95.1	97.0	97.3	97.9
DetGP (DWavg)	**93.4**	**95.2**	**96.3**	**97.5**	98.8	**94.3**	**96.2**	**97.7**	**98.1**	98.5

Table 1: AUC scores for link prediction on the *Cora* and *HepTh* dataset. The top four models only have structural embedding, while the rest use text information.

	Cora				DBLP			
%Training Nodes	**10%**	**30%**	**50%**	**70%**	**10%**	**30%**	**50%**	**70%**
LINE(Tang et al. 2015)	53.9	56.7	58.8	60.1	42.7	43.8	43.8	43.9
TADW (Yang et al. 2015)	71.0	71.4	75.9	77.2	67.6	68.9	69.2	69.5
CANE (Tu et al. 2017)	81.6	82.8	85.2	86.3	71.8	73.6	74.7	75.2
DMTE (Zhang et al. 2018a)	81.8	83.9	86.3	87.9	72.9	74.3	75.5	76.1
WANE (Shen et al. 2019)	81.9	83.9	86.4	88.1	NA	NA	NA	NA
DetGP (Wavg) only Text	78.1	81.2	84.7	85.3	71.4	73.3	74.2	74.9
DetGP (Wavg) only Struct	70.9	79.7	81.5	82.3	70.0	71.4	72.6	73.3
DetGP (Wavg)	80.5	85.4	86.7	88.5	76.9	78.3	79.1	79.3
DetGP (DWavg)	**83.1**	**87.2**	**88.2**	**89.8**	**78.0**	**79.3**	**79.6**	**79.8**

Table 2: Test Macro-F1 scores for multi-label node classification on *Cora* and *DBLP* dataset.

test on the Cora and DBLP datasets, which have group label information, where DetGP yields the best performance under all situations. This demonstrates that the proposed model can learn both representative textual and structural embeddings. The ablation study results (only textual embeddings *vs.* only structural embeddings) demonstrates that textual attributes are more important than edge connections in classification task. To describe the effect of learning the weighting in the diffusion, for the experiment on Cora with 10% nodes given for training, the learned weights in $\boldsymbol{\alpha}$ are $[0.58, 0.12, 0.24, 0.05]$. Thus, local and second order transition features are more important.

4.3 Dynamic Network Embedding

One of the advantages of the proposed model is the ability to quickly estimate embeddings on newly added nodes. To test the effectiveness of our model, we split nodes into training and testing sets. The embedding model is learned only from training nodes with corresponding edges. To evaluate, we embed the testing nodes directly without updating the model parameters. In this section we mainly focus on the performance of dynamic structural embeddings. Therefore, the text encoder is fixed as word embedding average (Wavg) for simplicity.

Previous works (Tu et al. 2017; Zhang et al. 2018a; Shen et al. 2019) on textual network embedding require the overall connection information to train the structural embedding, which cannot directly assign (without re-training) structural embeddings to a new coming node with connection information unknown during training. Therefore, the aforementioned methods cannot be applied to dynamic networks. To obtain comparable baselines to DetGP, we propose several strategies based on the idea of inductive graph representation learning (GraphSAGE) (Hamilton, Ying, and Leskovec 2017), which generates embedding for new nodes by aggregating the neighbors' embeddings. The two embedding assigning strategies are: (a) **Neighbor-Aggregate**: aggregating the *structural embeddings* from the neighbors in the training set, as the structural embedding for the new node; (b) **GraphSAGE**: aggregating the *textual embeddings* from the neighbors, then passing through a fully-connected layer to get the new node's structural embedding. For neighborhood information aggregating, we use the mean aggregator and the max-pooling aggregator as mentioned in (Hamilton, Ying, and Leskovec 2017).

We evaluate the dynamic embeddings for test nodes on link prediction and node classification tasks. For both tasks, we split the nodes into training and testing sets with different proportions (10%, 30%, 50%, 70%). When embedding new testing nodes, only their textual attributes and connections with existing training nodes are provided. For the link prediction task, we predict the edges between testing nodes

	Cora				HepTh			
%Training Nodes	**10%**	**30%**	**50%**	**70%**	**10%**	**30%**	**50%**	**70%**
Only Text (Wavg)	61.2	77.9	87.9	90.3	68.3	83.7	84.2	86.9
Neighbor-Aggregate (Max-Pooling)	54.6	69.1	78.7	87.3	59.6	78.3	79.9	80.7
Neighbor-Aggregate (Mean)	61.8	78.4	88.0	91.2	68.2	83.9	85.5	88.3
GraphSAGE (Max-Pooling)	62.1	78.6	88.6	92.4	68.4	85.8	88.1	91.2
GraphSAGE (Mean)	62.2	79.1	88.9	92.6	69.1	85.9	89.0	92.4
DetGP	**62.9**	**81.1**	**90.9**	**93.0**	**70.7**	**86.6**	**90.7**	**93.3**

Table 3: Test AUC scores for link prediction on *Cora* and *HepTh* datasets.

	Cora				DBLP			
% Training Nodes	**10%**	**30%**	**50%**	**70%**	**10%**	**30%**	**50%**	**70%**
Only Text (Wavg)	60.2	76.3	83.5	84.8	56.7	67.9	70.4	73.5
Neighbor-Aggregate (Max-Pooling)	55.8	70.2	78.4	80.5	51.8	60.5	68.3	70.6
Neighbor-Aggregate (Mean)	60.1	77.2	84.1	85.0	56.8	68.2	71.3	74.7
GraphSAGE (Max-Pooling)	61.3	78.2	85.1	86.3	58.9	69.1	72.4	74.9
GraphSAGE (Mean)	61.4	78.4	85.5	**86.6**	59.0	69.3	72.7	75.1
DetGP	**62.1**	**79.3**	**85.8**	**86.6**	**60.2**	**70.1**	**73.2**	**75.8**

Table 4: Test Macro-F1 scores for multi-label node classification on *Cora* and *DBLP* dataset.

based on the inner product between their node embeddings; for node classification, an SVM classifier is trained based on embeddings of training nodes. When new nodes come, we first embed the nodes using the trained model and then use the previously learned SVM to predict the label.

The results of link prediction and node classification are given in Tables 3 and 4, respectively. In both tasks, the Neighbor-Aggregate strategy with mean aggregator shows slight improvement to the baseline with only a text encoder. However, it does not work well with the max-pooling aggregator, implying that the unsupervised max-pooling on pre-trained neighbor structural embeddings cannot learn a good representation. The GraphSAGE strategies (with both mean and pooling aggregator) show notable improvements compared with Wavg and Neighbor-Aggregate. Unlike the unsupervised pooling, the GraphSAGE pooling aggregator is trained with a fully-connected layer on top, which shows comparable result to the mean aggregator. The proposed DetGP significantly outperforms other baselines, especially when the proportion of training set is small. A reasonable explanation is, when the training set is small, new nodes will have few connections with the training nodes, which causes high variance in the results of aggregating neighborhood embeddings. However, instead of aggregating, our DetGP infers the structural embedding via a Gaussian process with pre-learned inducing points, which is more robust than the information passed by neighbor nodes.

4.4 Inducing Points

Figure 3 gives the t-SNE visualization of the learned DetGP structural embeddings on the Cora citation dataset. The model is learned using all edges and all of the nodes with their textual information. We set the number of inducing points to $M = 20$. To avoid the computational instability caused by the inverse matrix $\boldsymbol{K}_{ZZ}^{-1}$, we update inducing points with a smaller learning rate, which is set to one-tenth of the learning rate for the text encoder. The inducing points $\boldsymbol{z}$ are visualized as red filled circles in Figure 3. Textual embeddings are plotted with 7 different colors, representing the 7 node classes. Note that the inducing points $\boldsymbol{z}$ fully cover the space of the categories, implying that the learned inducing points meaningfully cover the distribution of the textual embeddings.

Figure 3: t-SNE visualization of learned network structural embeddings of Cora. Unfilled circles are individual nodes and inducing points are denoted as red dots.

5 Conclusions

We propose a novel textual network embedding framework that learns representative node embeddings for static textual network, and also effectively adapts to dynamic graph structures. This is achieved by introducing a GP network structural embedding layer, that first maps each node to the inducing points, and then embeds them by taking advantage of the non-parametric representation. We also consider multiple hops to weight local and global graph structures. The graph structure is injected in the kernel matrix, where the kernel between two nodes use the whole graph information based on multiple hops. Our final embedding contains

both *structural* and *textual* information. Empirical results demonstrate the effectiveness of the proposed algorithm.

Acknowledgements: The research reported here was supported in part by DARPA, DOE, NIH, NSF and ONR.

References

Abu-El-Haija, S.; Perozzi, B.; Al-Rfou, R.; and Alemi, A. A. 2018. Watch your step: Learning node embeddings via graph attention. In *NeurIPS*, 9180–9190.

Airoldi, E. M.; Blei, D. M.; Fienberg, S. E.; and Xing, E. P. 2008. Mixed membership stochastic blockmodels. *JMLR* 1981–2014.

Cer, D.; Yang, Y.; Kong, S.-y.; Hua, N.; Limtiaco, N.; John, R. S.; Constant, N.; Guajardo-Cespedes, M.; Yuan, S.; Tar, C.; et al. 2018. Universal sentence encoder. *arXiv preprint arXiv:1803.11175*.

Du, L.; Wang, Y.; Song, G.; Lu, Z.; and Wang, J. 2018. Dynamic network embedding: An extended approach for skip-gram based network embedding. In *IJCAI*, 2086–2092.

Fan, W.; Ma, Y.; Li, Q.; He, Y.; Zhao, E.; Tang, J.; and Yin, D. 2019. Graph neural networks for social recommendation. *arXiv preprint arXiv:1902.07243*.

Goyal, P.; Kamra, N.; He, X.; and Liu, Y. 2018. Dyngem: Deep embedding method for dynamic graphs. *arXiv preprint arXiv:1805.11273*.

Grover, A., and Leskovec, J. 2016. node2vec: Scalable feature learning for networks. In *SIGKDD*, 855–864. ACM.

Guo, J.; Xu, L.; and Chen, E. 2018. Spine: structural identity preserved inductive network embedding. *arXiv preprint arXiv:1802.03984*.

Hamilton, W.; Ying, Z.; and Leskovec, J. 2017. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, 1024–1034.

Hanley, J. A., and McNeil, B. J. 1982. The meaning and use of the area under a receiver operating characteristic (roc) curve. *Radiology* 143(1):29–36.

Kingma, D. P., and Ba, J. 2015. Adam: A method for stochastic optimization. *ICLR*.

Kipf, T. N., and Welling, M. 2016. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308*.

Kipf, T. N., and Welling, M. 2017. Semi-supervised classification with graph convolutional networks. *ICLR*.

Leskovec, J.; Kleinberg, J.; and Faloutsos, C. 2007. Graph evolution: Densification and shrinking diameters. *TKDD*.

Li, J.; Dani, H.; Hu, X.; Tang, J.; Chang, Y.; and Liu, H. 2017. Attributed network embedding for learning in a dynamic environment. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, 387–396. ACM.

Li, J.; Cheng, K.; Wu, L.; and Liu, H. 2018. Streaming link prediction on dynamic attributed networks. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, 369–377. ACM.

Lin, Z.; Feng, M.; Santos, C. N. d.; Yu, M.; Xiang, B.; Zhou, B.; and Bengio, Y. 2017. A structured self-attentive sentence embedding. *ICLR*.

Ma, J.; Cui, P.; and Zhu, W. 2018. Depthlgp: learning embeddings of out-of-sample nodes in dynamic networks. In *AAAI*.

Melamud, O.; Goldberger, J.; and Dagan, I. 2016. context2vec: Learning generic context embedding with bidirectional lstm. In *SIGNLL*, 51–61.

Ng, Y. C.; Colombo, N.; and Silva, R. 2018. Bayesian semi-supervised learning with graph gaussian processes. In *NeurIPS*.

Perozzi, B.; Al-Rfou, R.; and Skiena, S. 2014. Deepwalk: Online learning of social representations. In *SIGKDD*, 701–710. ACM.

Powers, D. M. 2011. Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation. *Bioinfo Publications*.

Rasmussen, C. E. 2006. Gaussian processes for machine learning. In *Springer*.

Seo, Y.; Defferrard, M.; Vandergheynst, P.; and Bresson, X. 2018. Structured sequence modeling with graph convolutional recurrent networks. In *International Conference on Neural Information Processing*, 362–373. Springer.

Shen, D.; Zhang, X.; Henao, R.; and Carin, L. 2019. Improved semantic-aware network embedding with fine-grained word alignment. *EMNLP*.

Snelson, E., and Ghahramani, Z. 2006. Sparse gaussian processes using pseudo-inputs. In *Advances in neural information processing systems*, 1257–1264.

Sun, X.; Guo, J.; Ding, X.; and Liu, T. 2016. A general framework for content-enhanced network representation learning. *arXiv preprint arXiv:1610.02906*.

Tang, J.; Qu, M.; Wang, M.; Zhang, M.; Yan, J.; and Mei, Q. 2015. Line: Large-scale information network embedding. In *Proceedings of the 24th international conference on world wide web*, 1067–1077. International World Wide Web Conferences Steering Committee.

Titsias, M., and Lawrence, N. D. 2010. Bayesian gaussian process latent variable model. In *AISTATS*, 844–851.

Titsias, M. 2009. Variational learning of inducing variables in sparse gaussian processes. In *AISTATS*, 567–574.

Trivedi, R.; Dai, H.; Wang, Y.; and Song, L. 2017. Know-evolve: Deep temporal reasoning for dynamic knowledge graphs. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, 3462–3471. JMLR. org.

Trivedi, R.; Farajtabar, M.; Biswal, P.; and Zha, H. 2018. Representation learning over dynamic graphs. *arXiv preprint arXiv:1803.04051*.

Tu, C.; Liu, H.; Liu, Z.; and Sun, M. 2017. Cane: Context-aware network embedding for relation modeling. In *ACL*.

Yang, C.; Liu, Z.; Zhao, D.; Sun, M.; and Chang, E. 2015. Network representation learning with rich text information. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*.

Ying, R.; He, R.; Chen, K.; Eksombatchai, P.; Hamilton, W. L.; and Leskovec, J. 2018. Graph convolutional neural networks for web-scale recommender systems. In *SIGKDD*, 974–983. ACM.

Yu, K., and Chu, W. 2008. Gaussian process models for link analysis and transfer learning. In *NIPS*, 1657–1664.

Zhang, X.; Li, Y.; Shen, D.; and Carin, L. 2018a. Diffusion maps for textual network embedding. In *NeurIPS*.

Zhang, Y.; Yao, Q.; Shao, Y.; and Chen, L. 2018b. Nscaching: Simple and efficient negative sampling for knowledge graph embedding. *arXiv preprint arXiv:1812.06410*.

Zhou, L.; Yang, Y.; Ren, X.; Wu, F.; and Zhuang, Y. 2018. Dynamic network embedding by modeling triadic closure process. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Zitnik, M., and Leskovec, J. 2017. Predicting multicellular function through multi-layer tissue networks. *Bioinformatics* 33(14):i190–i198.