The Thirty-Second AAAI Conference
on Artificial Intelligence (AAAI-18)

DepthLGP: Learning Embeddings of Out-of-Sample Nodes in Dynamic Networks

Jianxin Ma, Peng Cui, Wenwu Zhu
Department of Computer Science and Technology, Tsinghua University, China
majx13fromthu@gmail.com, cuip@tsinghua.edu.cn, wwzhu@tsinghua.edu.cn

Abstract

Network embedding algorithms to date are primarily designed for static networks, where all nodes are known before learning. How to infer embeddings for out-of-sample nodes, i.e. nodes that arrive after learning, remains an open problem. The problem poses great challenges to existing methods, since the inferred embeddings should preserve intricate network properties such as high-order proximity, share similar characteristics (i.e. be of a homogeneous space) with in-sample node embeddings, and be of low computational cost. To overcome these challenges, we propose a Deeply Transformed High-order Laplacian Gaussian Process (DepthLGP) method to infer embeddings for out-of-sample nodes. DepthLGP combines the strength of nonparametric probabilistic modeling and deep learning. In particular, we design a high-order Laplacian Gaussian process (hLGP) to encode network properties, which permits fast and scalable inference. In order to further ensure homogeneity, we then employ a deep neural network to learn a nonlinear transformation from latent states of the hLGP to node embeddings. DepthLGP is general, in that it is applicable to embeddings learned by any network embedding algorithms. We theoretically prove the expressive power of DepthLGP, and conduct extensive experiments on real-world networks. Empirical results demonstrate that our approach can achieve significant performance gain over existing approaches.

Introduction

Network embedding (Perozzi, Al-Rfou, and Skiena 2014) automates the process of extracting low-dimensional feature vectors, termed embeddings, for nodes in a network. Despite the remarkable success they are achieving in tasks such as classification and recommendation, most network embedding algorithms in the literature to date are primarily designed for static networks, where all nodes are known before learning. However, for large-scale networks, it is infeasible to rerun network embedding whenever new nodes arrive, especially considering the fact that rerunning network embedding also results in the need of retraining all downstream classifiers. How to efficiently infer proper embeddings for out-of-sample nodes, i.e. nodes that arrive after the embedding process, remains largely unanswered.

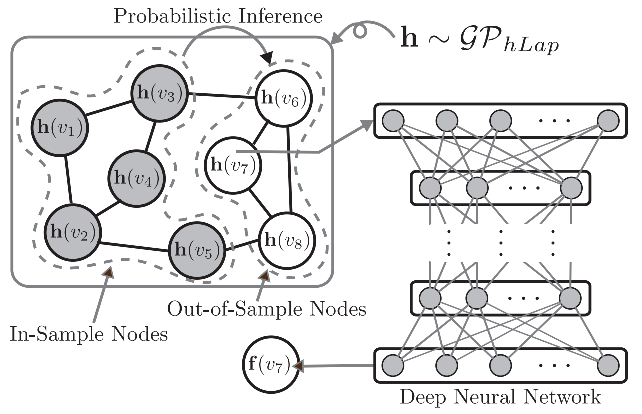

Figure 1: Here v_6, v_7, and v_8 are out-of-sample nodes. And $\mathbf{h}(\cdot)$ is a latent function. The values of the shaded nodes are learned during training. To predict $\mathbf{f}(v_7)$, DepthLGP first predicts $\mathbf{h}(v_7)$ via probabilistic inference, then passes $\mathbf{h}(v_7)$ through a neural network to obtain the final result.

Several graph-based methods in literature can be adjusted to infer out-of-sample embeddings given in-sample ones. Many of them deduce outcomes (embeddings) of new nodes by performing information propagation (Zhu and Ghahramani 2002), or optimizing a loss that encourages smooth changes between linked nodes (Zhu, Ghahramani, and Lafferty 2003; Delalleau, Bengio, and Roux 2005). There are also methods that aim to learn a mapping from node features to outcomes (embeddings), by imposing a manifold regularizer derived from the graph (Belkin, Niyogi, and Sindhwani 2006). The node features can be text attributes, or rows of the adjacency matrix when attributes are unavailable. With the learned mapping, the embeddings of out-of-sample nodes can then be predicted.

However, the recent progress of network embedding imposes new challenges to the existing methods. Firstly, the inferred embeddings of out-of-sample nodes should preserve intricate network properties with embeddings of in-sample nodes. For example, high-order proximity, among many other properties, is deemed particularly essential to be preserved by network embedding (Tang et al. 2015; Cao, Lu, and Xu 2015; Ou et al. 2016), and thus must be reflected by the inferred embeddings. Secondly, as downstream applications (e.g. classification) treat in-sample and out-of-sample nodes equally, the inferred embeddings and in-sample embeddings should possess similar characteristics (e.g. magnitude, mean, variance), i.e. be of a homogeneous space, resulting in the need of a model expressive enough to characterize the embedding space. Finally, maintaining high prediction speed is crucial, particularly when considering the highly dynamic nature of real-world networks. This final point is even more challenging due to the need for simultaneously fulfilling the previous two requirements.

The existing graph-based methods cannot address these challenges well. They are originally proposed to work on a constructed similarity graph (instead of a real-world network), and therefore consider primarily the basic smoothness requirement, while important network properties such as high-order proximity are mostly neglected. Furthermore, propagation/smoothness-based approaches typically use a naïve model, which might struggle with fitting the embedding space and can lead to poor homogeneity. On the other hand, regularization-based approaches (Belkin, Niyogi, and Sindhwani 2006) that may use a more advanced model (Tomar and Rose 2014) face an efficiency issue: their potentially lengthy training procedure cannot start before new nodes arrive, or they will be unable to utilize edges among new nodes (though edges between old and new nodes can be easily leveraged), as their prediction routine will not try to explore the subnetwork formed by new nodes.

In this paper, we propose a Deeply Transformed High-order Laplacian Gaussian Process (DepthLGP) method (see Figure 1) to infer out-of-sample embeddings. We combine the strength of nonparametric probabilistic modeling and deep neural networks. More specifically, we first design a high-order Laplacian Gaussian process (hLGP) prior with a carefully constructed kernel that encodes important network properties such as high-order proximity. Each node is associated with a latent state that follows the prior hLGP. We then employ a deep neural network to learn a nonlinear transformation function from these latent states to node embeddings. The introduction of a deep neural network increases the expressive power of our model and improves the homogeneity of inferred embeddings with in-sample embeddings. Theories on the expressive power of DepthLGP are derived. Overall, our method is fast and scalable. Training needs zero knowledge of out-of-sample nodes, and thus can be carried out in advance. The prediction routine revisits the evolved network rapidly and can produce inference results analytically with desirable time complexity linear to the number of in-sample nodes. DepthLGP is a general solution, in that it is applicable to embeddings learned by any network embedding algorithms. Extensive experiments on real-world networks further demonstrate that our method can achieve significant performance gain over existing approaches.

The DepthLGP Model

In this section, we first formally formulate the out-of-sample node problem, and then present the DepthLGP model as well as how to perform prediction and training, after which we derive theories on the expressive power of DepthLGP.

Problem Definition

We primarily consider undirected networks in this study. Let $\mathcal{G}$ be the set of all possible networks and $\mathcal{V}$ be the set of all possible nodes. Given a specific network $G = (V, E) \in \mathcal{G}$ with nodes $V = \{v_1, v_2, \ldots, v_n\} \subset \mathcal{V}$ and edges E, a network embedding algorithm aims to learn values of a function $\mathbf{f} : \mathcal{V} \to \mathbb{R}^d$ for nodes in V.

As the network evolves over time, a batch of m new nodes $V^* = \{v_{n+1}, v_{n+2}, \ldots, v_{n+m}\} \subset \mathcal{V} \backslash V$ arrives and expands G into a larger network $G' = (V', E')$. Here $V' = V \cup V^*$. Nodes in V^* are called out-of-sample nodes. Our problem, then, is to infer values of $\mathbf{f}(v)$ for $v \in V^*$, given $G' = (V', E')$ and $\mathbf{f}(v)$ for $v \in V$.

Model Description

DepthLGP first assumes that there exists a latent function $\mathbf{h} : \mathcal{V} \to \mathbb{R}^s$, and the embedding function $\mathbf{f} : \mathcal{V} \to \mathbb{R}^d$ is transformed from the said latent function. To be more specific, let $\mathbf{g} : \mathbb{R}^s \to \mathbb{R}^d$ be the transformation function. DepthLGP then assumes that $\mathbf{f}(v) = \mathbf{g}(\mathbf{h}(v))$ for all $v \in \mathcal{V}$. Since the transformation can potentially be highly nonlinear, we use a deep neural network to serve as $\mathbf{g}(\cdot)$. DepthLGP further assumes that the s output dimensions of $\mathbf{h}(\cdot)$, i.e. $h_k : \mathcal{V} \to \mathbb{R}$ for $k = 1, 2, \ldots, s$, can be modeled independently. In other words, we deal with each $h_k(v)$ of $\mathbf{h}(v) = [h_1(v), h_2(v), \ldots, h_s(v)]^\top$ separately.

Let us focus on $h_k(\cdot)$ for the moment. Each $h_k(\cdot)$ is associated with a kernel that measures the similarity between nodes of a network. Take $G' = (V', E')$ with $V' = \{v_1, v_2, \ldots, v_{n+m}\}$ for example, the said kernel produces a kernel matrix $\mathbf{K}_k \in \mathbb{R}^{(n+m)\times(n+m)}$ for G':

$$\begin{aligned}
\mathbf{K}_k &\triangleq \left[\mathbf{I} + \eta_k \mathbf{L}(\hat{\mathbf{A}}_k) + \zeta_k \mathbf{L}(\hat{\mathbf{A}}_k \hat{\mathbf{A}}_k)\right]^{-1}, \\
\hat{\mathbf{A}}_k &\triangleq \operatorname{diag}(\boldsymbol{\alpha}_k)\mathbf{A}'\operatorname{diag}(\boldsymbol{\alpha}_k), \\
\boldsymbol{\alpha}_k &\triangleq [a_{v_1}^{(k)}, a_{v_2}^{(k)}, \ldots, a_{v_{n+m}}^{(k)}]^\top,
\end{aligned}$$

where $\mathbf{A}'$ is the adjacency matrix of G'. And $\eta_k \in [0, \infty), \zeta_k \in [0, \infty)$ as well as $a_v^{(k)} \in [0, 1]$ for $v \in \mathcal{V}$ are parameters of the kernel. The function $\operatorname{diag}(\cdot)$ returns the diagonal matrix corresponding to its vector input. The function $\mathbf{L}(\cdot)$ treats its input as an adjacency matrix and returns the Laplacian matrix, i.e. $\mathbf{L}(\mathbf{A}) = \operatorname{diag}(\sum_i \mathbf{A}_{:,i}) - \mathbf{A}$. The definition of the kernel matrix involves inverting a large matrix. However, the matrix inversion can be avoided without approximation, as we will illustrate in the next subsection.

The parameters of the proposed kernel have clear physical meanings. The coefficient η_k indicates the strength of first-order proximity (i.e. two connected nodes are likely to be similar), while ζ_k is for second-order proximity (i.e. two nodes with common neighbors are likely to be similar). On the other hand, $a_v^{(k)}$ represents a node weight, i.e. how much attention we should pay to node v when performing prediction. The values of $a_v^{(k)}$ for in-sample nodes are learned along with η_k and ζ_k (as well as the parameters of the neural network $\mathbf{g}(\cdot)$) during training. The values of $a_v^{(k)}$ for out-of-sample nodes are set to one during prediction, since we are always interested in these new nodes when inferring embeddings for them. The node weights assist DepthLGP in spotting and avoiding uninformative nodes. For example, this design alleviates the harmful effect of bot users that follow many random people in a social network.

It can be shown that the proposed kernel matrix $\mathbf{K}_k$ is positive definite, hence a valid kernel matrix. The kernel can be seen as a generalization of the regularized Laplacian kernel (Smola and Kondor 2003), in that we further introduce node weighting and a second-order term. We term the proposed kernel the high-order Laplacian kernel.

DepthLGP assumes that each sub-function $h_k(\cdot)$ follows a zero-mean Gaussian process (GP) (Rasmussen and Williams 2005) parameterized by the high-order Laplacian kernel, i.e. $h_k \sim \mathcal{GP}^{(k)}_{hLap}$. In other words, for any $G_t = (V_t, E_t) \in \mathcal{G}$ with $V_t = \{v^{(t)}_1, v^{(t)}_2, \dots, v^{(t)}_{n_t}\} \subset \mathcal{V}$, we have:

$$[h_k(v^{(t)}_1), h_k(v^{(t)}_2), \dots, h_k(v^{(t)}_{n_t})]^\top \sim \mathcal{N}(\mathbf{0}, \mathbf{K}^{(t)}_k),$$

where $\mathbf{K}^{(t)}_k$ is the high-order Laplacian kernel matrix computed on G_t.

We can now summarize DepthLGP as follows:

$$\begin{aligned} &h_k \sim \mathcal{GP}^{(k)}_{hLap}, && k = 1, 2, \dots, s,\\ &\mathbf{h}(v) \triangleq [h_1(v), h_2(v), \dots, h_s(v)]^\top, && v \in \mathcal{V},\\ &\mathbf{f}(v) \mid \mathbf{h}(v) \sim \mathcal{N}(\mathbf{g}(\mathbf{h}(v)), \sigma^2\mathbf{I}), && v \in \mathcal{V}, \end{aligned}$$

where σ is a hyper-parameter to be manually specified. The neural network $\mathbf{g}(\cdot)$ is necessary here, since $\mathbf{f}(\cdot)$ itself might not follow the GP prior exactly. The introduction of $\mathbf{g}(\cdot)$ allows the model to fit $\mathbf{f}(\cdot)$ more precisely.

Prediction

Before new nodes arrive, we have the initial network $G = (V, E)$ with $V = \{v_1, v_2, \dots, v_n\}$. We also know the values of $\mathbf{f}(v)$ for $v \in V$. The prediction routine assumes that there is a training procedure conducted on G and $\mathbf{f}(v)$ for $v \in V$ before new nodes arrive. The training procedure learns $\eta_k, \zeta_k, h_k(v), a^{(k)}_v$ for $k = 1, 2, \dots, s$, $v \in V$, and the parameters of the transformation function $\mathbf{g}(\cdot)$.

As the network evolves over time, m new nodes $V^* = \{v_{n+1}, v_{n+2}, \dots, v_{n+m}\}$ arrive and G evolves into $G' = (V', E')$ with $V' = V \cup V^*$. The prediction routine aims to predict $\mathbf{f}(v)$ for $v \in V^*$ by maximizing $p(\{\mathbf{f}(v) : v \in V^*\} \mid \{\mathbf{f}(v) : v \in V\}, \{\mathbf{h}(v) : v \in V\})$, which, according to our model, is equal to:

$$p(\{\mathbf{f}(v) : v \in V^*\} \mid \{\mathbf{h}(v) : v \in V\}).$$

However, this requires integrating over all possible $\mathbf{h}(v)$ for $v \in V^*$. We hence approximate it by instead maximizing:

$$p(\{\mathbf{f}(v) : v \in V^*\}, \{\mathbf{h}(v) : v \in V^*\} \mid \{\mathbf{h}(v) : v \in V\}).$$

According to our model, it is equal to:

$$\begin{aligned} &p(\{\mathbf{f}(v) : v \in V^*\} \mid \{\mathbf{h}(v) : v \in V^*\})\\ &\quad \times p(\{\mathbf{h}(v) : v \in V^*\} \mid \{\mathbf{h}(v) : v \in V\}). \end{aligned}$$

Algorithm 1 The Prediction Routine

```
**Require:** $G' = (V', E')$    $\triangleright$ $G = (V, E)$ evolves into $G'$.
**Ensure:** predicted values of $\mathbf{f}(v)$, $v \in V^*$   $\triangleright$ $V^* \triangleq V' \setminus V$.
 1: $\triangleright$ Let $V = \{v_1, v_2, \dots, v_n\}$ be the old nodes.
 2: $\triangleright$ Let $V^* = \{v_{n+1}, v_{n+2}, \dots, v_{n+m}\}$ be the new nodes.
 3: $\triangleright$ Let $\mathbf{A}'$ be the adjacency matrix of $G'$.
 4: for $k = 1, 2, \dots, s$ do
 5:     $\triangleright$ Values of $a^{(k)}_v$ are set to 1 for $v \in V^*$.
 6:     $\boldsymbol{\alpha} \leftarrow [a^{(k)}_{v_1}, a^{(k)}_{v_2}, \dots, a^{(k)}_{v_{n+m}}]^\top$
 7:     $\hat{\mathbf{A}} \leftarrow \mathrm{diag}(\boldsymbol{\alpha})\ \mathbf{A}\ \mathrm{diag}(\boldsymbol{\alpha})$
 8:     $\triangleright$ Function $\mathbf{L}(\cdot)$ below treats $\hat{\mathbf{A}}$ and $\hat{\mathbf{A}}\hat{\mathbf{A}}$ as adjacency matrices, and returns their Laplacian matrices.
 9:     $\mathbf{M} \leftarrow \mathbf{I} + \eta_k \mathbf{L}(\hat{\mathbf{A}}) + \zeta_k \mathbf{L}(\hat{\mathbf{A}}\hat{\mathbf{A}})$
10:     $\mathbf{M}_{*,*} \leftarrow$ the bottom-right $m \times m$ block of $\mathbf{M}$
11:     $\mathbf{M}_{*,x} \leftarrow$ the bottom-left $m \times n$ block of $\mathbf{M}$
12:     $\triangleright$ Let $\mathbf{z}^{(k)}_x \triangleq [h_k(v_1), h_k(v_2), \dots, h_k(v_n)]^\top$.
13:     $\triangleright$ Compute $\mathbf{M}_{*,x}\mathbf{z}^{(k)}_x$ first below for efficiency.
14:     $\mathbf{z}^{(k)}_* \leftarrow -\mathbf{M}^{-1}_{*,*}\mathbf{M}_{*,x}\mathbf{z}^{(k)}_x$   $\triangleright$ $\mathbf{z}^{(k)}_*$ is a prediction of $[h_k(v_{n+1}), h_k(v_{n+2}), \dots, h_k(v_{n+m})]^\top$.
15: end for
16: for $v \in V^*$ do
17:     $\triangleright$ The previous lines have produced a prediction of $\mathbf{h}(v) = [h_1(v), h_2(v), \dots, h_s(v)]^\top$. The line below now uses the said prediction to further predict $\mathbf{f}(v)$.
18:     compute $\mathbf{g}(\mathbf{h}(v))$    $\triangleright$ It is a prediction of $\mathbf{f}(v)$.
19: end for
```

It can be maximized [1] by first maximizing the second term, i.e. $p(\mathbf{h}(v) : v \in V^*\} \mid \{\mathbf{h}(v) : v \in V\}) = \prod_{k=1}^{s} p(\{h_k(v) : v \in V^*\} \mid \{h_k(v) : v \in V\})$, and then setting $\mathbf{f}(v) = \mathbf{g}(\mathbf{h}(v))$ for $v \in V^*$.

Let us now focus on the subproblem of maximizing:

$$p(\{h_k(v) : v \in V^*\} \mid \{h_k(v) : v \in V\}). \qquad (1)$$

Since $h_k \sim \mathcal{GP}_{hLap}$, by definition we have:

$$\begin{aligned} [h_k(v_1), h_k(v_2), \dots, &h_k(v_n), h_k(v_{n+1}), \dots, h_k(v_{n+m})]^\top\\ &\sim \mathcal{N}(\mathbf{0}, \mathbf{K}_k), \end{aligned}$$

where $\mathbf{K}_k$ is the kernel matrix computed on G'. We then have the following result:

$$\begin{aligned} \mathbf{z}^{(k)}_* \mid \mathbf{z}^{(k)}_x &\sim \mathcal{N}(\mathbf{K}_{*,x}\mathbf{K}^{-1}_{x,x}\mathbf{z}^{(k)}_x, \mathbf{K}_{*,*} - \mathbf{K}_{*,x}\mathbf{K}^{-1}_{x,x}\mathbf{K}^\top_{*,x}),\\ \mathbf{z}^{(k)}_x &\triangleq [h_k(v_1), h_k(v_2), \dots, h_k(v_n)]^\top,\\ \mathbf{z}^{(k)}_* &\triangleq [h_k(v_{n+1}), h_k(v_{n+2}), \dots, h_k(v_{n+m})]^\top, \end{aligned}$$

where $\mathbf{K}_{x,x}$, $\mathbf{K}_{*,x}$, and $\mathbf{K}_{*,*}$ are the upper-left $n \times n$ block, the bottom-left $m \times n$ block, and the bottom-right $m \times m$ block of $\mathbf{K}_k$, respectively. Computing $\mathbf{K}_{*,x}\mathbf{K}^{-1}_{x,x}\mathbf{z}^{(k)}_x$ is expensive. However, it can be shown to be equivalent to

[1]The first term $p(\{\mathbf{f}(v) : v \in V^*\} \mid \{\mathbf{h}(v) : v \in V^*\})$ is maximized with $\mathbf{f}(v) = \mathbf{g}(\mathbf{h}(v))$, and the maximum value of this probability density is a *constant* independent of $\mathbf{h}(v)$. Hence we can focus on maximizing the second term first.

$-\mathbf{M}_{*,*}^{-1}\mathbf{M}_{*,x}\mathbf{z}_x^{(k)}$, where $\mathbf{M}_{*,x}$ and $\mathbf{M}_{*,*}$ are the bottom-left $m \times n$ block and the bottom-right $m \times m$ block of $\mathbf{K}_k^{-1}$, respectively. Obtaining $\mathbf{K}_k^{-1}$ is cheap, because the matrix inversion gets cancelled out. Computing $\mathbf{M}_{*,*}^{-1}$ is much easier, since $m \ll n$. As a result, Equation 1 is maximized with:

$$\mathbf{z}_*^{(k)} = -\mathbf{M}_{*,*}^{-1}\mathbf{M}_{*,x}\mathbf{z}_x^{(k)}.$$

As a side node, maximizing Equation 1 is equivalent to minimizing the following criterion:

$$\sum_{u\in V'} [h_k(u)]^2 + \\ \frac{1}{2}\eta_k \sum_{u,v\in V'} a_u^{(k)} A'_{uv} a_v^{(k)} [h_k(u) - h_k(v)]^2 + \\ \frac{1}{2}\zeta_k \sum_{u,v,w\in V'} a_u^{(k)} A'_{uw} a_w^{(k)} a_w^{(k)} A'_{wv} a_v^{(k)} [h_k(u) - h_k(v)]^2,$$

where A'_{uv} is the weight of the edge between u and v. This alternative formula illustrates the physical meaning of η, ζ and $a_v^{(k)}$, from a different perspective.

The prediction routine is summarized in Algorithm 1.

Training

Training is conducted on the initial network $G = (V, E)$ with the values of $\mathbf{f}(v)$ for $v \in V$. Since it does not depend on the evolved network $G' = (V', E')$, it can be carried out before new nodes arrive. It aims to find suitable parameters of the neural network $\mathbf{g}(\cdot)$ and proper values of $\eta_k, \zeta_k, a_v^{(k)}, h_k(v)$ for $v \in V$ and $k = 1, 2, \dots, s$.

We train the model with empirical risk minimization (ERM). The idea of training a probabilistic model with ERM has been explored before by many researchers, e.g. (Stoyanov, Ropson, and Eisner 2011). Using ERM training here eliminates the need to specify σ, and is faster and more scalable as it avoids computing determinants.

The basic idea of the training procedure (Algorithm 2) is to first sample some subgraphs from G, then treat a small portion of nodes in each subgraph as if they were out-of-sample nodes, and minimize the empirical risk on these training samples. For each sample $G'_t = (V'_t, E'_t)$, we first sample a small set of seed nodes V_t^* from G along a random walk path. We then sample a set of nodes V_t from the neighborhood of V_t^*. The neighborhood of V_t^* consists of the nodes that are no more than two steps away from V_t^*. Let $V'_t = V_t^* \cup V_t$. The subgraph G'_t is then induced in G by V'_t. The nodes in V_t^* are treated as out-of-sample nodes.

We use Adam (Kingma and Ba 2015), a method similar to stochastic gradient descent, for optimizing the parameters. Gradients are computed using back-propagation (Rumelhart, Hinton, and Williams 1988; Dreyfus 1962).

On the Expressive Power of DepthLGP

The first theorem below demonstrates the expressive power of DepthLGP, while the second theorem emphasizes the importance of second-order proximity.

Algorithm 2 The Training Routine

```
Require: $G = (V, E)$; $\mathbf{f}(v)$ for $v \in V$
Ensure: $\eta_k, \zeta_k, a_v^{(k)}, h_k(v)$ for $v \in V$ and $k = 1, 2, \dots, s$;
    parameters of the neural network $\mathbf{g}(\cdot)$
 1: for $t = 1, 2, \dots, T$ do
 2:     $V_t^* \leftarrow$ a few nodes sampled along a random walk
 3:     $V_t \leftarrow$ some nodes in $V_t^*$'s neighborhood
 4:     $V'_t \leftarrow V_t \cup V_t^*$
 5:     $G'_t \leftarrow$ the subgraph induced in $G$ by $V'_t$
 6:     Execute Algorithm 1, but using $G'_t$ in place of $G'$,
        $V_t$ in place of old nodes, and $V_t^*$ in place of new nodes.
        Save its prediction of $\mathbf{f}(v)$ as $\tilde{\mathbf{f}}(v)$ for $v \in V_t^*$.
 7:     loss $\leftarrow \frac{1}{|V_t^*|} \sum_{v\in V_t^*} \|\mathbf{f}(v) - \tilde{\mathbf{f}}(v)\|_{l^2}^2$
 8:     Use back-propagation to compute the gradient of the
        loss with respect to $\eta_k, \zeta_k, a_v^{(k)}, h_k(v)$ for $v \in V_t$, and
        the parameters of $\mathbf{g}(\cdot)$.
 9:     Apply gradient descent.
10: end for
```

Theorem 1 (Expressive Power). *For any $\epsilon > 0$, any nontrivial $G = (V, E)$ and any $\mathbf{f} : \mathcal{V} \to \mathbb{R}^d$, there exists a parameter setting for DepthLGP, such that: for any $v^* \in V$, after deleting all information (except G) related with v^*, DepthLGP can still recover $\mathbf{f}(v^*)$ with error less than ϵ, by treating v^* as a new node and using Algorithm 1 on G.*

Remark. A nontrivial G means that all connected components of G have at least three nodes. The information related with v^* includes $\mathbf{f}(v^*)$, $h_k(v^*)$ and $a_{v^*}^{(k)}$ for $k = 1, 2, \dots, s$. During prediction, $a_{v^*}^{(k)}$ is set to one, since v^* is treated as a new node. The error is expressed in terms of l^2-norm. The proof is a constructive proof based on the universal approximation property of neural networks (Cybenko 1989; Hornik 1991) (see supplement material).

Theorem 2 (On Second-Order Proximity). *Theorem 1 will not hold if DepthLGP does not model second-order proximity. That is, there will exist $G = (V, E)$ and $\mathbf{f} : \mathcal{V} \to \mathbb{R}^d$ that DepthLGP cannot model, if ζ_k is fixed to zero.*

Remark. It can be verified by finding a counterexample (see supplement material).

Extensions and Variants

Integrating into an Embedding Algorithm

DepthLGP can be incorporated into an existing network embedding algorithm to derive a new network embedding algorithm capable of handling out-of-sample nodes. Take node2vec (Grover and Leskovec 2016) for example. The objective of node2vec can be abstracted into:

$$\min_{\theta,\mathbf{F}} \mathcal{L}_\theta(\mathbf{F}, G).$$

The columns of $\mathbf{F} \in \mathbb{R}^{d\times n}$ are the embeddings to be learned, and θ contains other parameters.

We now define a function $\mathbf{f}_\phi : \mathcal{V} \to \mathbb{R}^d$ parameterized by ϕ. For each $v \in V$, it first samples nodes from the neighborhood of v and induces the subgraph that contains v and

Table 1: The ten research areas selected from DBLP.

Research Field	Conference
Database	ICDE, VLDB, SIGMOD
Data Mining	KDD, ICDM, SDM, CIKM
Information Retrieval	SIGIR
Artificial Intelligence	IJCAI, AAAI, ICML, NIPS
Computer Vision	CVPR, ICCV
Theory	STOC, SODA, COLT
Computational Linguistics	ACL, EMNLP, COLING
Computer Networks	SIGCOMM, INFOCOM
Operating Systems	SOSP, OSDI
Programming Languages	POPL

the sampled nodes. It treats v as an out-of-sample node and the nodes from the neighborhood as in-sample nodes. It then runs Algorithm 1 on the subgraph to obtain the embedding of v, which is the value of $\mathbf{f}_\phi(v)$. By definition, ϕ contains parameters of a neural network, η_k, ζ_k, $a_v^{(k)}$, and $h_k(v)$ for $v \in V$, $k = 1, 2, \ldots, s$. To derive a new embedding algorithm, which we name node2vec++, we replace $\mathbf{F}$ with $\mathbf{f}_\phi(\cdot)$:

$$\min_{\theta,\phi} \mathcal{L}_\theta([\mathbf{f}_\phi(v_1), \mathbf{f}_\phi(v_2), \ldots, \mathbf{f}_\phi(v_n)], G).$$

Efficient Variants

Handling new nodes in a batch-by-batch manner is more memory-saving when the number of new nodes is large. For each unprocessed new node v, we first find the largest connected component that contains v and consists of new nodes only. Let V_t^* be the connected component. We then sample a set of old nodes V_t from the neighborhood of V_t^*. Finally, we run the prediction routine on the subgraph induced by $V_t^* \cup V_t$ to obtain embeddings for the new nodes in V_t^*. We can repeat this procedure until all new nodes are processed.

Some simplifications can be made in order to allow a more efficient implementation. In particular, sharing node weights across different dimensions hurts little for most embedding algorithms. Though theoretically it will reduce the expressive power. Additionally, we can keep $\eta_1 = \ldots = \eta_s$ and $\zeta_1 = \ldots = \zeta_s$ when the network embedding algorithm uses the same objective to learn the different dimensions.

Empirical Results

Experiment Settings

We conduct experiments on the datasets listed below.

- DBLP: We extract a co-author network from dblp.org, consisting of ten research areas (see Table 1). We treat the research areas as labels. We take the snapshot at the end of 2015 as the initial network (59,277 nodes, 205,804 edges), and the snapshot at the end of 2016 as the evolved network (4,944 new nodes, 25,612 new edges).
- PPI (Breitkreutz et al. 2008): A protein-protein interaction network. We use the largest connected component (3,852 nodes, 37,841 edges, 50 labels) extracted from the version provided by (Grover and Leskovec 2016). We sample 256 nodes with degree less than sixteen along a random walk to serve as out-of-sample nodes, as the network contains no timestamp.
- BlogCatalog (Tang and Liu 2009): An online social network (10,312 users, 333,983 edges, 39 labels). Again, we sample 256 nodes to serve as out-of-sample nodes.

We compare DepthLGP to the baselines listed below.

- LocalAvg: This baseline produces an embedding for an out-of-sample node by averaging the embeddings of neighboring in-sample nodes.
- Manifold Regularization (MRG): This baseline is a manifold regularized neural network (Tomar and Rose 2014). It does not perform well when trained solely on the initial network. We therefore train it on the evolved network for fair comparison, even though it is impractical to do so in a production environment (due to lengthy optimization). We found that initializing the weights of the first layer with in-sample node embeddings and using l^1-normalized rows of the adjacency matrix as features greatly improve its convergence speed.
- LabelProp: Recent propagation/smoothness-based methods are mostly proposed for classification. A small portion of them can be adjusted to solve our problem (a multivariate regression problem). We found that the more recent methods (that can be adjusted to solve the regression problem) do not beat the vanilla version of label propagation (Zhu and Ghahramani 2002) when it comes to our problem. We hence report performance of the vanilla version (Zhu and Ghahramani 2002).

We first run an embedding algorithm (node2vec, LINE, or GraRep) on the initial network to obtain 128-dimensional embeddings for in-sample nodes. We then train DepthLGP on the initial network. Finally, we run DepthLGP's prediction routine, as well as our baselines, on the evolved network to produce embeddings for out-of-sample nodes.

We use $\mathbf{g}(\mathbf{x}) = \mathbf{x} + \tilde{\mathbf{g}}(\mathbf{x})$ for DepthLGP, where $\tilde{\mathbf{g}}(\mathbf{x})$ is a neural network with a single hidden layer of 64 units. We use LeakyReLU as the activation function. This choice of $\mathbf{g}(\cdot)$ resembles a residual network (He et al. 2016). It allows us to initialize values of $\mathbf{h}(v)$ with values of $\mathbf{f}(v)$ for in-sample nodes. We set the number of seed nodes to be four when sampling a subgraph for training. We ensure that the sampled neighborhood of the seed nodes contains at most 512 one-step nodes, and further restrict the number of sampled two-step nodes to be less than a half of the number of sampled one-step nodes. We find this setting provides good convergence speed, based on our experience on a development dataset (a tenth of the DBLP network).

We repeat the experiments for ten times and report averaged performance. Additionally, we rerun the embedding algorithm on the evolved network to obtain ground-truth embeddings for new nodes, and report their performance (titled Upper Bound in Table 2 and Table 3). Ideally we want DepthLGP's performance to be as close to these upper bounds as possible. To illustrate the importance of ensuring homogeneity, we also report performance of hLGP, which is mostly the same as DepthLGP, except that it uses $\mathbf{g}(\mathbf{x}) = \mathbf{x}$ instead of a neural network.

Table 2: Quality of the inferred out-of-sample embeddings in terms of their performance in multi-label classification. hLGP is a simplified version of DepthLGP, in that it removes the neural transformation.

			Baselines			This Work		Upper Bound
Metric	Embedding	Network	LocalAvg	MRG	LabelProp	hLGP	DepthLGP	(rerunning)
Macro-F1(%)	LINE	DBLP	37.89	42.15	40.83	47.33	**48.25**	(49.07)
		PPI	10.52	10.02	12.42	13.42	**13.72**	(13.91)
		BlogCatalog	13.25	11.30	17.07	17.41	**18.03**	(18.90)
	GraRep	DBLP	50.61	55.79	55.02	57.43	**58.67**	(62.92)
		PPI	13.65	13.75	12.38	14.80	**14.84**	(15.33)
		BlogCatalog	14.76	14.80	14.71	15.94	**18.45**	(20.15)
	node2vec	DBLP	53.83	59.34	59.25	60.89	**62.63**	(64.87)
		PPI	15.05	13.43	13.78	15.85	**16.54**	(16.81)
		BlogCatalog	15.10	14.04	19.16	19.77	**20.32**	(20.82)
Micro-F1(%)	LINE	DBLP	49.58	50.49	50.88	54.01	**54.94**	(55.84)
		PPI	18.10	15.71	18.81	20.71	**21.42**	(21.43)
		BlogCatalog	27.40	23.21	30.79	31.36	**31.90**	(32.20)
	GraRep	DBLP	60.17	60.62	60.48	61.44	**62.29**	(65.44)
		PPI	20.23	20.35	20.23	20.79	**21.44**	(21.88)
		BlogCatalog	36.44	30.79	33.90	37.57	**38.14**	(38.37)
	node2vec	DBLP	60.54	62.29	62.52	62.83	**64.56**	(65.63)
		PPI	19.70	18.25	18.25	22.63	**23.11**	(23.41)
		BlogCatalog	34.83	25.82	36.94	37.96	**39.64**	(40.34)

Multi-label Classification

We first measure the quality of the inferred out-of-sample embeddings, in terms of their performance in multi-label classification. We use the in-sample embeddings (as features) and their corresponding labels to train a one-vs-all logistic regression classifier, and then use the classifier to predict the labels of the out-of-sample nodes given the inferred out-of-sample embeddings. We report Macro-F1 and Micro-F1 scores in Table 2.

DepthLGP significantly outperforms all the baselines consistently. And the margin between DepthLGP and rerunning the whole network embedding algorithm is close enough for DepthLGP to be useful in practice. The fact that hLGP outperforms all the other baselines indicates the importance of preserving properties such as high-order proximity, while the fact that hLGP is weaker than DepthLGP suggests the necessity of ensuring homogeneity.

Link Prediction

We then measure the quality of the inferred embeddings when it comes to link prediction. We hide half of the new edges (they serve as positive links to be predicted) when inferring out-of-sample embeddings, and report the area under the receiver operating characteristic curve (AUC) (see Table 3). Again, DepthLGP significantly outperforms the baselines and achieves closer performance to the upper bounds.

Computational Efficiency

We assess the efficiency of the prediction routine with a performance-versus-time analysis, conducted on PPI (see Figure 2). We observe that DepthLGP can indeed produce high-performing embeddings in a very short period of time.

Integrating into node2vec

We derive node2vec++, a new embedding algorithm based on DepthLGP and node2vec, and assess its performance on DBLP. node2vec++ can predict embeddings for out-of-sample nodes by itself. We use DepthLGP to assist node2vec in inferring out-of-sample embeddings. The results (Table 4) suggest that node2vec++ produces comparable in-sample embeddings and infers better out-of-sample embeddings.

Related Work

Network Embedding. After DeepWalk (Perozzi, Al-Rfou, and Skiena 2014), several network embedding algorithms are proposed to better characterize network properties such as high-order proximity (Tang et al. 2015; Cao, Lu, and Xu 2015), nonlinearity (Wang, Cui, and Zhu 2016), neighborhood (Grover and Leskovec 2016), non-transitivity (Ou et al. 2015), and community structure (Wang et al. 2017b). Embedding special networks are also explored (Ou et al. 2016; Zheng and Skillicorn 2015; Wang et al. 2017a; Yang et al. 2015; Dong, Chawla, and Swami 2017). Yet, few of them can deal with out-of-sample nodes. For example, factorization-based embedding algorithms (Cao, Lu, and Xu 2015; Ou et al. 2016; Wang et al. 2017b) are inherently crippled when it comes to out-of-sample nodes, since new nodes imply dimension expansion of matrices to be factorized and necessitates retraining. On the other hand, though some algorithms (Tang et al. 2015; Wang, Cui, and Zhu 2016) can quickly infer embeddings for a subset of new nodes based on low-order proximity, they can only handle new nodes directly connected with the embedded part, as they do not explore edges among new nodes.

Table 3: Quality of the inferred out-of-sample embeddings in terms of their performance in link prediction.

Metric	Embedding	Network	Baselines: LocalAvg	Baselines: MRG	Baselines: LabelProp	This Work: hLGP	This Work: DepthLGP	Upper Bound (rerunning)
AUC(%)	LINE	DBLP	72.87	72.87	77.39	80.63	**81.18**	(82.33)
		PPI	52.34	51.78	52.77	57.04	**60.45**	(60.57)
		BlogCatalog	55.51	51.01	54.71	54.74	**55.53**	(55.76)
	GraRep	DBLP	84.15	85.88	86.32	87.25	**87.40**	(91.95)
		PPI	62.80	68.55	66.48	67.60	**68.85**	(69.61)
		BlogCatalog	45.60	41.24	47.29	47.42	**48.11**	(48.26)
	node2vec	DBLP	68.49	76.90	77.98	81.36	**82.54**	(89.02)
		PPI	38.90	40.54	46.79	53.16	**55.37**	(59.74)
		BlogCatalog	54.65	38.41	55.40	55.43	**55.47**	(55.86)

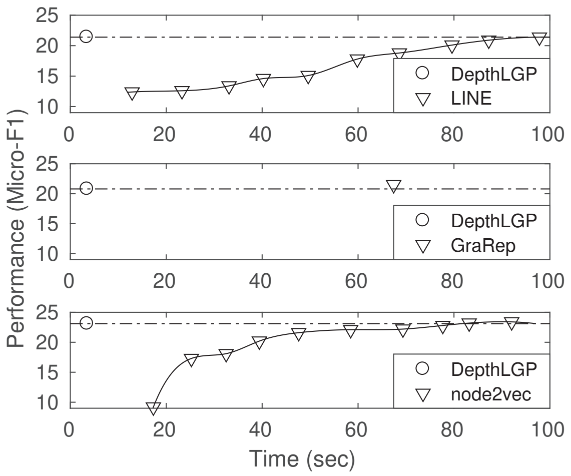

Figure 2: Performance versus time. DepthLGP and GraRep are represented by points, as they are closed-form solvers. node2vec and LINE iteratively learn embeddings of new nodes, we hence report their performance every fixed number of iterations. DepthLGP can infer high-performing embeddings in much shorter time.

Learning on Graphs. Several methods originated from graph-based semi-supervised learning (GSSL) (Chapelle, Schlkopf, and Zien 2010; Zhu 2005) can serve as baselines to solve the out-of-sample node problem (see section "Introduction" on limitations of these methods). Some of these methods take a propagation approach (Szummer and Jaakkola 2001; Zhu and Ghahramani 2002), while the others optimize a loss that encourages smoothness (Zhu, Ghahramani, and Lafferty 2003; Joachims 2003; Delalleau, Bengio, and Roux 2005; Subramanya and Bilmes 2008). There are also methods based on regularizing a traditional predictor with a graph (Belkin, Niyogi, and Sindhwani 2006; Karlen et al. 2008; Chen, Tsang, and Xu 2012; Tomar and Rose 2014). Historically, these methods focus primarily on classification and only a portion of them can be adjusted to solve our problem (a multivariate regression problem).

Table 4: Performance of node2vec++, a new embedding algorithm derived by integrating DepthLGP into node2vec.

	Metric	node2vec	node2vec++
In-Sample	Macro-F1(%)	64.87	65.17
	Micro-F1(%)	65.63	65.61
	AUC(%)	90.02	89.20
Out-of-Sample	Macro-F1(%)	62.63	63.88
	Micro-F1(%)	64.56	64.60
	AUC(%)	82.54	85.88

Gaussian Processes. To the extent of our knowledge, (Chu et al. 2006; Yu et al. 2006; Yu and Chu 2007; Silva, Chu, and Ghahramani 2007) pioneer the application of Gaussian processes (Rasmussen and Williams 2005) to learning on networks. They target various tasks, such as supervised learning, semi-supervised learning, link prediction, and transfer learning. Our problem and model are drastically different from theirs. As far as we know, we are the first to use a GP model for network embedding. GP models are demonstrated to be effective in working with network data, mainly because they can model uncertainty exhibited in both observed and unobserved edges and provide high-quality generalization on unseen nodes.

Conclusion

In this work, we propose DepthLGP to infer embeddings for out-of-sample nodes in an effective and efficient manner. DepthLGP combines the strength of nonparametric probabilistic modeling and deep learning. In the future, we hope to extend our work to special networks, or apply it to solve similar problems such as out-of-vocabulary words.

Acknowledgements

This work is supported by the National Program on Key Basic Research Project (No. 2015CB352300), and the National Natural Science Foundation of China (No. 61772304, No. 61521002, No. 61531006, and No. U1611461). Thanks for the research fund of Tsinghua-Tencent Joint Laboratory for Internet Innovation Technology, and the Young Elite Scientist Sponsorship Program by CAST.

References

Belkin, M.; Niyogi, P.; and Sindhwani, V. 2006. Manifold regularization: A geometric framework for learning from labeled and unlabeled examples. *JMLR*.

Breitkreutz, B.-J.; Stark, C.; Reguly, T.; Boucher, L.; Breitkreutz, A.; Livstone, M.; Oughtred, R.; Lackner, D. H.; Bhler, J.; Wood, V.; Dolinski, K.; and Tyers, M. 2008. The biogrid interaction database: 2008 update. *Nucleic Acids Research*.

Cao, S.; Lu, W.; and Xu, Q. 2015. Grarep: Learning graph representations with global structural information. In *Proceedings of CIKM 2015*.

Chapelle, O.; Schlkopf, B.; and Zien, A. 2010. *Semi-Supervised Learning*. The MIT Press.

Chen, L.; Tsang, I. W.; and Xu, D. 2012. Laplacian embedded regression for scalable manifold regularization. *IEEE Transactions on Neural Networks and Learning Systems*.

Chu, W.; Sindhwani, V.; Ghahramani, Z.; and Keerthi, S. S. 2006. Relational learning with gaussian processes. In *Proceedings of NIPS 2006*.

Cybenko, G. 1989. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*.

Delalleau, O.; Bengio, Y.; and Roux, N. L. 2005. Efficient non-parametric function induction in semi-supervised learning. In *Proceedings of AISTATS 2005*.

Dong, Y.; Chawla, N. V.; and Swami, A. 2017. Metapath2vec: Scalable representation learning for heterogeneous networks. In *Proceedings of KDD 2017*.

Dreyfus, S. 1962. The numerical solution of variational problems. *Journal of Mathematical Analysis and Applications*.

Grover, A., and Leskovec, J. 2016. Node2vec: Scalable feature learning for networks. In *Proceedings of KDD 2016*.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *Proceedings of CVPR 2016*.

Hornik, K. 1991. Approximation capabilities of multilayer feedforward networks. *Neural Networks*.

Joachims, T. 2003. Transductive learning via spectral graph partitioning. In *Proceedings of ICML 2003*.

Karlen, M.; Weston, J.; Erkan, A.; and Collobert, R. 2008. Large scale manifold transduction. In *Proceedings of ICML 2008*.

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

Ou, M.; Cui, P.; Wang, F.; Wang, J.; and Zhu, W. 2015. Non-transitive hashing with latent similarity components. In *Proceedings of KDD 2015*.

Ou, M.; Cui, P.; Pei, J.; Zhang, Z.; and Zhu, W. 2016. Asymmetric transitivity preserving graph embedding. In *Proceedings of KDD 2016*.

Perozzi, B.; Al-Rfou, R.; and Skiena, S. 2014. Deepwalk: Online learning of social representations. In *Proceedings of KDD 2014*.

Rasmussen, C. E., and Williams, C. K. I. 2005. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press.

Rumelhart, D. E.; Hinton, G. E.; and Williams, R. J. 1988. Learning representations by back-propagating errors. In *Neurocomputing: Foundations of Research*. The MIT Press.

Silva, R.; Chu, W.; and Ghahramani, Z. 2007. Hidden common cause relations in relational learning. In *Proceedings of NIPS 2007*.

Smola, A. J., and Kondor, R. 2003. Kernels and regularization on graphs. In *Proceedings of COLT 2003*.

Stoyanov, V.; Ropson, A.; and Eisner, J. 2011. Empirical risk minimization of graphical model parameters given approximate inference, decoding, and model structure. In *Proceedings of AISTATS 2011*.

Subramanya, A., and Bilmes, J. 2008. Soft-supervised learning for text classification. In *Proceedings of EMNLP 2008*.

Szummer, M., and Jaakkola, T. 2001. Partially labeled classification with markov random walks. In *Proceedings of NIPS 2001*.

Tang, L., and Liu, H. 2009. Relational learning via latent social dimensions. In *Proceedings of KDD 2009*.

Tang, J.; Qu, M.; Wang, M.; Zhang, M.; Yan, J.; and Mei, Q. 2015. Line: Large-scale information network embedding. In *Proceedings of WWW 2015*.

Tomar, V. S., and Rose, R. C. 2014. Manifold regularized deep neural networks. In *Proceedings of INTERSPEECH 2014*.

Wang, S.; Tang, J.; Aggarwal, C.; Chang, Y.; and Liu, H. 2017a. Signed network embedding in social media. In *Proceedings of SDM 2017*.

Wang, X.; Cui, P.; Wang, J.; Pei, J.; Zhu, W.; and Yang, S. 2017b. Community preserving network embedding. In *Proceedings of AAAI 2017*.

Wang, D.; Cui, P.; and Zhu, W. 2016. Structural deep network embedding. In *Proceedings of KDD 2016*.

Yang, C.; Liu, Z.; Zhao, D.; Sun, M.; and Chang, E. Y. 2015. Network representation learning with rich text information. In *Proceedings of IJCAI 2015*.

Yu, K., and Chu, W. 2007. Gaussian process models for link analysis and transfer learning. In *Proceedings of NIPS 2007*.

Yu, K.; Chu, W.; Yu, S.; Tresp, V.; and Xu, Z. 2006. Stochastic relational models for discriminative link prediction. In *Proceedings of NIPS 2006*.

Zheng, Q., and Skillicorn, D. 2015. Spectral embedding of signed networks. In *Proceedings of SDM 2015*.

Zhu, X., and Ghahramani, Z. 2002. Learning from labeled and unlabeled data with label propagation. Technical report, Carnegie Mellon University.

Zhu, X.; Ghahramani, Z.; and Lafferty, J. 2003. Semi-supervised learning using gaussian fields and harmonic functions. In *Proceedings of ICML 2003*.

Zhu, X. 2005. *Semi-supervised Learning with Graphs*. Ph.D. Dissertation, Carnegie Mellon University.