Self-Paced Network Embedding

Hongchang Gao
Department of Electrical and Computer Engineering
University of Pittsburgh
Pittsburgh, USA
hongchanggao@gmail.com

Heng Huang*
Department of Electrical and Computer Engineering
University of Pittsburgh
Pittsburgh, USA
heng.huang@pitt.edu

ABSTRACT

Network embedding has attracted increasing attention in recent data mining research with many real-world applications. Network embedding is to learn low-dimensional representations for nodes in a network. A popular kind of existing methods, such as DeepWalk, Node2Vec, and LINE, learn node representations by pushing positive context node to the anchor node while pushing negative context nodes away from it in the low-dimensional vector space. When sampling the negative context nodes, they usually employ a predefined sampling distribution based on the node popularity. However, this sampling distribution often fails to capture the real informativeness of each node and cannot reflect the training state. To address these important problems, in this paper, we propose a novel self-paced network embedding method. Specifically, our method can adaptively capture the informativeness of each node based on the current training state, and sample negative context nodes in terms of their informativeness. The proposed self-paced sampling strategy can gradually select difficult negative context nodes with training process going on to learn better node representations. Moreover, to better capture the node informativeness for learning node representations, we extend our method to the generative adversarial network framework, which has the larger capacity to discover node informativeness. The extensive experiments have been conducted on the benchmark network datasets to validate the effectiveness of our proposed methods.

KEYWORDS

Network Embedding, Informativeness, Self-Paced Sampling Strategy

ACM Reference Format:
Hongchang Gao and Heng Huang. 2018. Self-Paced Network Embedding. In *KDD '18: The 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, August 19–23, 2018, London, United Kingdom.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3219819.3220041

*To whom all correspondence should be addressed. This work was partially supported by U.S. NIH R01 AG049371, NSF IIS 1302675, IIS 1344152, DBI 1356628, IIS 1619308, IIS 1633753.

KDD '18, August 19–23, 2018, London, United Kingdom

ACM ISBN 978-1-4503-5552-0/18/08…$15.00
https://doi.org/10.1145/3219819.3220041

1 INTRODUCTION

In recent years, network embedding has attracted a surge of attention because it is a fundamental tool for network data analysis. Network embedding aims at learning a low-dimensional representation for nodes in a network such that the downstream tasks, such as node classification, link prediction, and network visualization, can benefit from it. The essential idea to learn the low-dimensional representations is to preserve the proximity in the topological structure of a network. By preserving such proximities, nodes with large proximity will have similar low-dimensional representations, benefiting downstream tasks.

To obtain the effective low-dimensional representation, a wide variety of approaches have been proposed in recent years. They aim at capturing various proximities in a network. For example, the seminal DeepWalk model [21] employs random walk to obtain the node sequences and then reformulates network embedding as word embedding by regarding node sequences as word sequences. In this way, the proximity between nodes can be captured by Skip-gram [20] model. Afterwards, Node2Vec [6] was proposed based on the similar idea but with a novel random walk algorithm. Additionally, LINE [25] was introduced to preserve the first and second order proximity when learning the low-dimensional representation. GraRep [4] aims at preserving high order proximity during representation learning. Struc2Vec [22] targets to preserve to structural proximity by identifying structure identities.

Among these existing methods, the methods based on Skip-gram [20] model, such as DeepWalk [21], Node2Vec [6], and LINE [25] have attracted much attention. Their basic idea is to push together similar nodes while pushing away dissimilar nodes. Mathematically, it is formulated as a binary classification problem which classifies two similar nodes as a positive pair while classifying two dissimilar nodes as a negative pair. Although these approaches have shown good performance in many tasks, yet it has a drawback. Specifically, to train this binary classification problem, we should feed both positive and negative pairs to the model. For each node, its paired positive node can be easily obtained according to edges of the network [25] or the result of random walk [6, 21]. But how to obtain the negative nodes is not obvious. Some negative nodes are more informative while some are not. If employing random sampling method, we cannot discriminate the informativeness of different nodes.

Therefore, existing methods, such as DeepWalk [21], Node2Vec [6], and LINE [25], propose to sample negative nodes based on a pre-designed distribution. This distribution depends on the degree of nodes. The intuition behind this sampling method is that oversampling more connected nodes will lead to better performance because these nodes carry more information than less connected

ones. Although this strategy has correct intuition, the idea is incomplete because this method ignores that the less connected nodes may also be informative in practice. Thus, we cannot just use the connection information to determine the informativeness of a node. How to discover the really informative nodes for training models is important and also challenging.

Moreover, the sampling distribution employed by the existing network embedding methods [6, 21, 25] is static, which means that the sampling distribution does not change during training. As a result, the informativeness of a node is constant when training models. However, with the training process going on, some negative nodes are well pushed away from the anchor node while some are not. In such a case, we should put more focus on these difficult nodes. Therefore, it is better to sample nodes with different probabilities at different training phases. On the other hand, we cannot always focus on the difficult nodes, ignoring the easy ones. According to the learning process of human beings, we should start with easy samples when learning a new model and then learn difficult samples gradually [2, 13]. The reason is that we can learn a draft model by using easy samples and then refine it by taking difficult samples. In our case, if we always select difficult negative nodes to train our model, the easy ones cannot be fully utilized, and it is easy to disturb the low-dimensional representations by too much focus on the difficult nodes.

To address the above challenging problems, in this paper, we propose a novel self-paced network embedding method. With this method, we can dynamically sample the informative negative nodes for training models. Specifically, we propose a self-paced node sampling strategy. This strategy can discover the informativeness of each node based on current model parameters and then sample negative nodes according to their informativeness. In addition, this self-paced strategy can sample difficult negative nodes gradually with the training process going on. Moreover, we extend this self-paced sampling strategy to the generative adversarial network framework. The extensive experiments are conducted on seven benchmark network datasets to validate the effectiveness of our proposed methods.

2 RELATED WORK

2.1 Network Embedding

Network embedding has become a popular data mining research topic in recent years due to its important role in network data analysis. It is to learn a low-dimensional representation for each node in a network while preserving proximity. Recently, there has been much progress towards effective network embedding algorithms. For example, DeepWalk [21] was proposed to utilize random walk to get node sequences as word sentences, then employ Skip-gram [20] model to learn node embedding, which is essential to predict the context of each anchor node by maximizing the likelihood function as follows:

$$\max \prod_{i=1}^{n} \prod_{j \in c_i} p(v_j|v_i), \tag{1}$$

where v_i is the anchor node, v_j denotes its context node, and c_i denotes the context set of node v_i. $p(v_j|v_i)$ is the conditional probability of node v_j given node v_i. Based on this schema, LINE [25] and Node2Vec [6] were proposed respectively with different context constructing approaches. In addition, some other kinds of methods also make much progress. For example, GraRep [4] aims at preserving high order proximity during representation learning. Struc2Vec [22] aims to preserve structural proximity by identifying structure identities. M-NMF [29] captures community structure when learning the low-dimensional representation. TADW [30] and ANE [8] target to integrate the topological structure and node attributes to learn node representations. A dynamic network embedding method [15] was proposed to handle the dynamic networks. SDNE [27] employs deep autoencoder to capture the high non-linearity in the network. In [12], a semi-supervised network embedding method was proposed based on the graph convolutional network. Although these methods are promising in many tasks, in this paper, we will limit our interest to solve the model related to Eq. (1).

2.2 Sample Selection

Negative Sampling Negative sampling [7, 20] is an efficient approach for solving multi-class classification problems. In detail, the multi-class classification problem, such as problem (1), is considerably inefficient when the number of classes is very large. Negative sampling (or termed as contrastive learning) reformulates it as a binary classification problem by constructing positive pairs and negative pairs. The positive pair is usually easy to construct based on some prior knowledge. For example, positive pairs in a network can be easily constructed based on the connection between nodes [25]. The challenge is how to construct negative pairs. In [20], the authors propose to sample negative pairs based on the word popularity, while [25] considers constructing the sampling distribution in terms of the degree of nodes. Actually, both of them share a common intuition that more popular or connected samples should be selected more frequently since they are more informative. Under the context of recommender systems, [31] proposes a dynamic negative sampling algorithm based on the temporary recommend result. Recently, IRGAN [28] was proposed to have the same spirit as negative sampling for information retrieval. Unlike conventional negative sampling, it employs a generator to generate the negative samples dynamically.

Self-Paced Learning In machine learning community, how to select training samples to learn a good model is an important topic. As we know, human beings usually learn from easy concepts to complex ones. To mimic this cognitive activity of humans, curriculum learning [2] and self-paced learning [13] have attracted much attention. Particularly, based on some fixed prior knowledge, curriculum learning constructs a ranking function to assign different learning priorities to different samples. Self-paced learning selects training samples voluntarily by measuring the performance of samples dynamically as follows:

$$\min_{v_i, w} \sum_{i=1}^{n} v_i l(y_i, f(x_i; w)) - \lambda \sum_{i=1}^{n} v_i \,, \tag{2}$$

where w is the model parameter, $v_i \in \{0, 1\}$ indicates whether the training sample (x_i, y_i) is selected, and $l(\cdot, \cdot)$ denotes the loss function.

Along this line, many works have been proposed for different situations. For example, [9] proposes a self-paced learning algorithm

to select easy and diverse samples. [16] proposes the self-paced co-training under the semi-supervised settings. [24] is proposed for object tracking. [10] is proposed for combining curriculum learning and self-paced learning. However, as far as we know, there are no existing works for our settings. Thus, inspired by self-paced learning, we will focus on how to perform negative sampling from easy samples to difficult ones dynamically to solve Eq. (1).

3 NEW SELF-PACED NETWORK EMBEDDING

3.1 Problem Definition

Given a network $\mathcal{G} = \{\mathcal{V}, E\}$ where $\mathcal{V} = \{v_i\}_{i=1}^n$ denotes a set of n nodes and $E = [e_{ij}] \in \Re^{n \times n}$ denotes the adjacency matrix, if there exists an edge between node v_i and v_j, $e_{ij} = 1$. Otherwise, $e_{ij} = 0$. The embedding of each node v_i is a low-dimension vector $u_i \in \Re^d$.

Network embedding is to learn low-dimensional representations $\{u_i\}_{i=1}^n$ of nodes. The essential idea is to push similar nodes together while pushing dissimilar nodes away to each other in the low-dimensional vector space. Thus, DeepWalk [21], LINE [25], and Node2Vec [6] propose to optimize Eq. (1). By optimizing this model, the context node v_j of the anchor node v_i will have a high probability $p(v_j|v_i)$, while other nodes have a small $p(v_j|v_i)$. Thus, Eq. (1) can push positive context nodes v_j to the anchor node v_i while pushing negative context nodes away from the anchor node.

However, Eq. (1) actually is a multi-class classification problem in which the number of classes equals the number of nodes. When n is very large, it is inefficient. Therefore, DeepWalk [21], LINE [25], and Node2Vec [6] reformulate Eq. (1) as an equivalent problem and employ negative sampling method to accelerate it as follows:

$$\max \log p(v_p|v_i) + \sum_{j \in \mathcal{N}_{v_i}} \log(1 - p(v_j|v_i)), \tag{3}$$

where v_p denotes the positive context of node v_i while v_j denotes the negative context, $\mathcal{N}_{v_i}$ denotes the sampled negative context node set and $|\mathcal{N}_{v_i}| = k$. By using this effective negative sampling method, the complexity is reduced to $O(km)$ while the original complexity is $O(nm)$ where $O(m)$ is the complexity of computing the loss of each sample and $k \ll n$. Now, a natural question is how to select the negative context samples $\mathcal{N}_{v_i}$ effectively? In existing works [6, 21, 25], they use a predefined distribution, which reflects the popularity of nodes, to sample them. Specifically, LINE [25] constructs the sampling distribution based on the number of degrees of nodes such that a more connected node will be selected in a large probability. However, it has some drawbacks in practice as follows:

- The popularity-based sampling method cannot really reflect the informativeness of a node. Some less connected nodes can also be much informative in practice.
- The informativeness of a node is usually changing with the training process going on. But the predefined sampling method fails to reflect this change.

Therefore, in this paper, we will focus on developing effective negative sampling algorithms to address these problems.

3.2 Self-Paced Network Embedding

In this section, we will introduce a novel dynamic sampling method for network embedding: self-paced network embedding. At first, to fully capture the informativeness of nodes and reflect the training state, we propose a dynamic negative sampling method. Based on this method, we further propose a self-paced sampling method to feed difficult samples gradually with the training process going on to learn a better embedding result. At last, we extend the proposed self-paced sampling method to the generative adversarial network framework.

3.2.1 Informativeness of Nodes. Formally, the conditional probability in Eq. (1) and Eq. (3) is defined as follows:

$$p(v_j|v_i) = \sigma(u_j^T u_i) = \frac{1}{1 + \exp(-u_j^T u_i)}, \tag{4}$$

where u_i and u_j are the low-dimensional representation of node v_i and v_j respectively. Here, we call v_i *anchor node*. As we can see from this formulation, a large inner product between u_i and u_j denotes a high probability of v_j given v_i. For all the negative context nodes $\mathcal{N}_{v_i}$ of the anchor node v_i, they will have different conditional probabilities based on the currently learned low-dimensional representation. Moreover, as for a negative context node, if its conditional probability is high, it should be a *difficult negative context node* since it is close to the anchor point based on the inner product but we should push it away from the anchor node. Otherwise, it is an *easy negative context node* because it is far away from the anchor point. Thus, it is natural to use Eq. (4) to denote the *informativeness* of a negative context node given the anchor node. If $p(v_j|v_i)$ is large, v_j is more informative to the anchor node v_i. Otherwise, it is less informative since it has already been far away from the anchor point.

On the other hand, the essentiality of network embedding is to push away negative context nodes from the anchor point. Thus, to learn a good low-dimensional representation such that similar nodes are clustered while dissimilar nodes are separated, it is better to put more focus on difficult negative context nodes to push them away from the anchor node. Based on this intuition, it is natural to construct the negative sampling distribution based on the informativeness of nodes between the currently learned low-dimensional representation at each iteration. Mathematically, we define the informativeness-aware negative sampling distribution as follows:

$$p_{ij} = \frac{\exp(u_j^T u_i)}{\sum_{j \in \mathcal{N}_{v_i}} \exp(u_j^T u_i)}, \tag{5}$$

where $\mathcal{N}v_i$ is the negative context node set of node v_i. As shown in this formulation, if a negative context node is close to the anchor node, the inner product between them will be large so that the probability p_{ij} is large. As a result, the difficult negative context node has a large chance to be selected. Then, it will be updated more frequently than the easy ones so that it is pushed away from the anchor point. Thus, our method can always select the more informative nodes to update model parameters. On the contrary, the conventional sampling distribution used in [6, 21, 25] only focuses on the more connected nodes. With such a connection-based sampling method, even when a more connected node has

already been well separated from the anchor node, it still has a large chance to be updated. While a less connected node which is close to the anchor node has small chance to be pushed away from the anchor node. Thus, their result is not satisfactory. As for our method, we put more focus on the difficult negative context node. As a result, even though a negative context node is less connected, it still has a large chance to be selected if it has a large inner product with the anchor node. All in all, our proposed method can effectively select the more informative negative context node.

Furthermore, unlike the predefined sampling distribution employed in [6, 21, 25], our proposed sampling distribution can reflect the training state since the low-dimensional representation is updated at each iteration. In particular, the sampling distribution changes dynamically according to the current state. An easy negative context node may become a difficult one after updating model parameters, then it will have a large chance to be selected. As a conclusion, our proposed method can automatically and dynamically select the informative negative context node based on the training state.

3.2.2 Self-Paced Negative Sampling. Although the proposed sampling distribution in Eq. (5) can always select the more informative nodes at each iteration, yet it fails to utilize all nodes. Specifically, at each iteration, it always selects the difficult negative context node, the easy one almost has no chance to be pushed away from the anchor node. Although the easy negative context node may already be separated with the anchor node, yet it still needs to refine so that we can get a better result.

On the other hand, according to self-paced learning [2, 13], like human beings learning from easy samples to complex ones, we should feed easy samples at early training stage and gradually provide difficult samples with the training process going on. However, the sampling distribution in Eq. (5) always feed difficult samples at any training stage. Therefore, to fully utilize all nodes and gradually select difficult nodes, we propose the self-paced network embedding (SeedNE) model as follows:

$$\begin{aligned} &\max \log p(v_p|v_i) + \sum_{j \overset{p'_{ij}}{\sim} \mathcal{N}_{v_i}} \log(1 - p(v_j|v_i)) + l(\mu) \\ &s.t. \quad p'_{ij} = \begin{cases} p_{ij}, & p_{ij} < l(\mu) \\ 0, & otherwise \end{cases}, \end{aligned} \tag{6}$$

where $\mathcal{N}v_i$ is the negative context node set of node v_i, $l(\mu)$ is a threshold function parameterized by μ, which acts as a threshold to the sampling probability. In detail, for the probability p_{ij} obtained from Eq. (5), if it is larger than the threshold $l(\mu)$, we set it as zero as shown in Eq. (6). Actually, it is to set the probability of difficult negative context nodes to zero such that the easy ones have larger chance to be selected at the early training stage. With the training process going on, $l(\mu)$ is increasing so that difficult negative context nodes will be included gradually. This is consistent with our motivation. The concrete threshold function $l(\mu)$ is deferred to Section 4.

By optimizing problem (6) for each anchor node v_i, we can gradually sample difficult negative context nodes based on the training performance to learn low-dimensional representations of nodes. At last, we summarize the self-paced network embedding (SeedNE) method in Algorithm 1.

Algorithm 1 Self-Paced Network Embedding (SeedNE)

1: **for** each node v_i **do**
2: Sample positive context nodes v_p.
3: Sample negative context nodes v_j according to p'_{ij}.
4: Update node representations u_i, u_p, u_j and the parameter μ by SGD.
5: Update the sampling probability p'_{ij} as Eq. (5) and Eq. (6).
6: **end for**

3.2.3 Extension: Adversarial Self-Paced Network Embedding. In recent years, Generative Adversarial Network (GAN) [5] has attracted a surge of attention due to its flexibility to simulate distributions. The GAN framework includes a generator and a discriminator. The generator is to approximate the data distribution while the discriminator is to discriminate the approximated and true data distribution. Mathematically, the GAN framework is to optimize the following problem:

$$\min_{\theta} \max_{\phi} E_{x \sim p(x)} \log[D(x)] + E_{x \sim G(z), z \sim p(z)}[\log(1 - D(x))] , \tag{7}$$

where $D(\cdot)$ denotes the discriminator parameterized by ϕ, $G(\cdot)$ denotes the generator parameterized by θ, $p(x)$ denotes the data distribution, and $p(z)$ denotes a prior distribution. Here, the generator $G(\cdot)$ tries to use a deep neural network to map a simple prior distribution $p(z)$ to the complicated data distribution. By optimizing this objective function, the generator is expected to generate similar samples with true ones as well as possible.

Inspired by the GAN framework, we can use a generator to generate negative samples for Eq. (3). Specifically, for each anchor node v_i, it is defined as follows:

$$\begin{aligned} \min_{\theta} \max_{\phi} & E_{v_j \sim p_d(v_j|v_i)} \log p_\phi(v_j|v_i) \\ & + E_{v_j \sim G_\theta(v_j|v_i)}[\log(1 - p_\phi(v_j|v_i))] , \end{aligned} \tag{8}$$

where $p_\phi(v_j|v_i)$ is identical with Eq. (4) in which $\phi = \{u_i, u_j\}$, $p_d(v_j|v_i)$ denotes the sampling distribution for the positive context node of node v_i, $G_\theta(v_j|v_i)$ denotes the sampling distribution for the negative context node, which is constructed from the generator.

This model includes two components: discriminator and generator. The discriminator has the same functionality as Eq. (3), which pushes similar nodes together while pushing away different nodes in the low-dimensional vector space. The generator is to generate negative context nodes by constructing the sampling distribution $G_\theta(v_j|v_i)$. More details are explained in the following.

Discriminator Specifically, the discriminator is to solve the following problem:

$$\max_{\phi} E_{v_j \sim p_d(v_j|v_i)} \log p_\phi(v_j|v_i) + E_{v_j \sim G_\theta(v_j|v_i)}[\log(1 - p_\phi(v_j|v_i))] , \tag{9}$$

which is actually identical with Eq. (3). The only difference is how to obtain negative context nodes. The negative context node of this model is generated from the generator $G_\theta(v_j|v_i)$ while Eq. (3) uses a predefined distribution for sampling them. Similarly, the difference between this discriminator and Eq. (6) is also the sampling method.

Eq. (6) constructs the basic sampling distribution according to the current training state as shown in Eq. (5). This discriminator also sample negative nodes based on the current training state, but it uses a generator to construct such a distribution in a more flexible way, which will be shown in the following. By optimizing this model, we can obtain the embedding result $\phi = \{u_i\}_{i=1}^n$.

Generator On the other hand, the generator is to solve the following problem:

$$\min_{\theta} E_{v_j \sim G_\theta(v_j|v_i)}[\log(1 - p_\phi(v_j|v_i))] , \tag{10}$$

where the generator model $G_\theta(v_j|v_i)$ constructs a sampling distribution as follows:

$$G_\theta(v_j|v_i) = \frac{\exp(u_j'^T u_i')}{\sum_j \exp(u_j'^T u_i')} , \tag{11}$$

where $\theta = \{u_i'\}_{i=1}^n$ is the model parameter of the generator. Specifically, u_i' denotes the low-dimensional representation of node v_i in the generator. At first glance, Eq. (11) has no difference with Eq. (5). However, Eq. (5) uses the current embedding result $\{u_i\}_{i=1}^n$ to construct the sampling distribution while Eq. (11) employs a more flexible way to construct the sampling distribution to find the underlying informative negative context nodes. In particular, it constructs the sampling distribution by training another model parameterized by new parameters $\theta = \{u_i'\}_{i=1}^n$. With these new parameters, the sampling distribution will have more capacity to discover the informative nodes. Furthermore, similar with Eq. (5), these new parameters can also reflect the current embedding result. Specifically, the loss function in Eq. (10) is computed based on the current embedding result. Thus, when updating new model parameters $\theta = \{u_i'\}_{i=1}^n$ by gradient descent method, the current embedding result can directly affect θ through the gradient information.

With this flexible distribution, we can sample v_j as the negative context node of node v_i to train this model. However, there is a challenge to optimize Eq. (10). In particular, to use gradient descent method to update model parameter θ, the model should be continuous. However, the sample v_j that the loss function depends on is not the direct output of the generator $G_\theta(v_j|v_i)$. Instead, v_j is sampled from a node set according to $G_\theta(v_j|v_i)$, which is discrete. Thus, we cannot use gradient descent method directly to update model parameter θ. A practical way is to employ the policy gradient [23] to update model parameter θ, which is defined as follows:

$$\begin{aligned}
&\nabla_\theta E_{v_j \sim G_\theta(v_j|v_i)}[\log(1 - p_\phi(v_j|v_i))] \\
&= \sum_{j=1}^{K} \nabla_\theta G_\theta(v_j|v_i) \log(1 - p_\phi(v_j|v_i)) \\
&= \sum_{j=1}^{K} G_\theta(v_j|v_i) \nabla_\theta \log[G_\theta(v_j|v_i)] \log(1 - p_\phi(v_j|v_i)) \\
&= E_{v_j \sim G_\theta(v_j|v_i)} \nabla_\theta \log[G_\theta(v_j|v_i)] \log(1 - p_\phi(v_j|v_i)) .
\end{aligned} \tag{12}$$

However, take a look at the sampling distribution Eq. (11) defined in the generator, we can find that this distribution has the same spirit as Eq. (5), which always selects the difficult negative context node. Thus, it potentially ignores the easy negative context node and cannot gradually select difficult ones with the training process going on. To alleviate these problems, we further propose the adversarial self-paced network embedding (ASeedNE) method as follows:

$$\begin{aligned}
&\min_{\theta} \max_{\phi} E_{v_j \sim p_d(v_j|v_i)} \log p_\phi(v_j|v_i) + l(\mu) \\
&\qquad + E_{v_j \sim G'_\theta(v_j|v_i)}[\log(1 - p_\phi(v_j|v_i))] \\
&s.t. \quad G'_\theta(v_j|v_i) = \begin{cases} G_\theta(v_j|v_i), & G_\theta(v_j|v_i) < l(\mu) \\ 0, & otherwise . \end{cases}
\end{aligned} \tag{13}$$

There are still two components in this novel model. Specifically, the discriminator is to optimize the following problem:

$$\begin{aligned}
&\max_{\phi} E_{v_j \sim p_d(v_j|v_i)} \log p_\phi(v_j|v_i) + l(\mu) \\
&\quad + E_{v_j \sim G'_\theta(v_j|v_i)}[\log(1 - p_\phi(v_j|v_i))] \\
&s.t. \quad G'_\theta(v_j|v_i) = \begin{cases} G_\theta(v_j|v_i), & G_\theta(v_j|v_i) < l(\mu) \\ 0, & otherwise . \end{cases}
\end{aligned} \tag{14}$$

The difference between this discriminator and that in Eq. (9) is that here we employ a self-paced sampling strategy such that the difficult negative context node is gradually utilized to train this model. While the model in Eq. (9) always choose the most difficult negative context nodes to train modal parameters.

Additionally, the generator is to optimize the following problem:

$$\begin{aligned}
&\min_{\theta} E_{v_j \sim G'_\theta(v_j|v_i)}[\log(1 - p_\phi(v_j|v_i))] \\
&s.t. \quad G'_\theta(v_j|v_i) = \begin{cases} G_\theta(v_j|v_i), & G_\theta(v_j|v_i) < l(\mu) \\ 0, & otherwise . \end{cases}
\end{aligned} \tag{15}$$

For this generator, after sampling v_j according to the self-pace sampling strategy, we can still use the policy gradient defined in Eq. (12) to update model parameters.

At last, our adversarial self-paced network embedding method is summarized in Algorithm 2.

Algorithm 2 Adversarial Self-Paced Network Embedding (ASeedNE)

1: **repeat**
2: Sampling positive and negative context nodes according to p_d and G'_θ respectively to update the discriminator in Eq. (14).
3: Sampling negative context nodes according to G'_θ to update the generator in Eq. (15).
4: Update the sampling probability G'_θ according to Eq. (11) and Eq. (15).
5: **until** Converges

Note that although IRGAN [28] also utilizes the GAN framework to mimic the negative sampling procedure, yet our method is different with IRGAN. In particular, IRGAN always chooses the difficult negative samples while our method can gradually select difficult negative context nodes according to the training state. In this way, our method learns node representations from easy negative context nodes to get a draft of the embedding and then refine the embedding result by selecting difficult negative context nodes gradually.

3.2.4 Summarization. Compared with the predefined negative sampling distribution used in [6, 21, 25], our proposed SeedNE can effectively capture the informativeness of each node based on the current embedding result. Based on the informativeness of nodes, SeedNE can gradually select difficult negative context nodes to train the model by employing the self-paced sampling strategy. For the ASeedNE method, it can also discover the informativeness of each node in a more flexible way by training a generator. Similarly, the sampling distribution constructed by the generator can also reflect the current embedding result. All in all, both SeedNE and ASeedNE gradually select difficult negative context nodes to train model parameters.

4 EXPERIMENTS

In this section, we will describe our experiments to show the performance of the proposed methods.

4.1 Dataset Descriptions

In this paper, we use 7 benchmark datasets including social networks, citation networks, biomedical networks, and so on. The details about these datasets are shown as follows.

- Wikipedia (Wiki) [18]: This is a co-occurrence network of words in the Wikipedia database. Here, we employ the preprocessed version of [6]. Specifically, this network contains 4177 nodes, 184,812 edges. Additionally, there are 40 node labels which denote the Part-of-Speech tag.
- Protein-Protein Interactions (PPI) [3]: This is a biomedical network. We utilize the preprocessed network in [6]. The details about the preprocessing operation can be found in [6]. In particular, there are 3,890 nodes and 76,584 edges in this network. The number of node classes is 50. Different classes denote different biological states.
- Cora [19]: This is an academic citation network, which contains 2,708 machine learning papers. There are 5,429 edges in this network. These papers are from 7 classes, which represent the topic of these papers.
- Citeseer [19]: This is another citation network. It has 3,312 publications from 6 classes. The edge between two publications denotes the citation relationship between them. There are 4,660 edges totally in this network.
- BlogCatalog [26]: This is a social network from the BlogCatalog website. The nodes are bloggers and the edges are friendship relationships among bloggers. The class is inferred from bloggers' interests. Specifically, there are 10,312 nodes, 333,983 edges. The number of classes is 39.
- Facebook [14]: This is a social network where nodes represent users and edges denote the friendship between two users. This dataset is used for the link prediction task
- GR-QC [14]: This is a collaboration network where nodes denote authors and edges represent the collaborative relationship between two authors. This dataset is also used for the link prediction task.

we summarize the statistics of these benchmark datasets in Table 1.

Table 1: Description of Benchmark Datasets

Dataset	#Nodes	#Edges	#Labels
Wiki	4,777	184,812	40
PPI	3,890	76,584	50
Cora	2,708	5,278	7
Citeseer	3,312	4,660	6
BlogCatalog	10,312	333,983	39
Facebook	4,039	88,234	-
GR-QC	5,242	14,496	-

4.2 Baseline Methods

To show the performance of our proposed methods, we compare them with four state-of-the-art methods, including DeepWalk, Node2Vec, GraRep, LINE. The details about these methods are described as follows.

- DeepWalk [21]: This method utilizes random walk to get node sequences. By viewing node sequences as word sentences, DeepWalk formulates network embedding as word embedding so that it uses Skip-gram [20] model to learn node representations.
- Node2Vec [6]: Similar with DeepWalk, this method also uses Skip-gram [20] model to learn node representations. But it uses a biased random walk algorithm to get node sequences for constructing node contexts.
- GraRep [4]: This method proposes to preserve high order proximity by constructing k-step probability transition matrix when learning node representations.
- LINE [25]: This method proposes first and second order proximity among nodes. It learns to represent the node by preserving the first and second order proximity.

4.3 Experiment Settings

In our experiments, we set the window size as 10, the walk length as 80, and the number of walks as 10 for DeepWalk. For Node2Vec, it has the same settings as DeepWalk. Additionally, we set the parameter $p = 1$ and $q = 1$ for Node2Vec. For GraRep, the maximum matrix transition step is set as 5. For LINE, we set the number of negative samples as 5. To have a fair comparison, the dimension of node representations is set as 100 for all methods.

For our proposed methods, there is no any preprocess operation on the network. For each node, we use its connected neighbor nodes as the positive context nodes while the disconnected ones as the negative context nodes. When sampling negative context nodes, we set the number of negative context nodes as 1, that is $\mathcal{N}_{v_i} = 1$. The implementation is based on Tensorflow [1]. AdamSGD [11] algorithm is employed to update model parameters. The batch size is set as 100. Furthermore, the threshold function employed in Eq. (6) and Eq. (13) is defined as follows:

$$l(u) = \max(au^2 + b, 1)\,, \tag{16}$$

where $a > 0$ and $b > 0$ are set as different values for different datasets. Additionally, u is initialized as 1 for all datasets. We can see that this function is always positive and no larger than 1. When

Table 2: Node Classification Result of Wiki

Method	50%		70%		90%	
	Micro-F1	Macro-F1	Micro-F1	Macro-F1	Micro-F1	Macro-F1
DeepWalk	0.4720	0.0993	0.4850	0.1047	0.4795	0.0946
Node2Vec	0.4753	0.1058	0.4860	0.1154	0.5015	0.1056
GraRep	0.4729	0.1071	0.4929	0.1213	0.4839	0.1000
LINE	0.4830	0.1082	0.4919	0.1014	0.4868	0.1067
SeedNE	**0.5283**	**0.1207**	**0.5429**	**0.1428**	**0.5367**	**0.1355**
ASeedNE	**0.5343**	**0.1222**	**0.5449**	**0.1619**	**0.5513**	**0.1445**

Table 3: Node Classification Result of PPI

Method	50%		70%		90%	
	Micro-F1	Macro-F1	Micro-F1	Macro-F1	Micro-F1	Macro-F1
DeepWalk	0.1915	0.1629	0.2156	0.1853	0.2009	0.1659
Node2Vec	0.1858	0.1590	0.2004	0.1663	0.2040	0.1630
GraRep	0.2035	0.1730	0.1999	0.1712	0.2100	0.1654
LINE	0.2092	0.1812	0.2065	0.1768	0.2040	0.1675
SeedNE	**0.2191**	**0.1825**	**0.2268**	**0.1940**	**0.2131**	**0.1835**
ASeedNE	**0.2209**	**0.1854**	**0.2293**	**0.1925**	**0.2268**	**0.2010**

Table 4: Node Classification Result of Cora

Method	50%		70%		90%	
	Micro-F1	Macro-F1	Micro-F1	Macro-F1	Micro-F1	Macro-F1
DeepWalk	0.8111	0.8023	0.8069	0.8018	0.8044	0.7956
Node2Vec	0.8096	0.8034	0.8007	0.7989	0.8081	0.8080
GraRep	0.7749	0.7577	0.7872	0.7721	0.7897	0.7839
LINE	0.8118	0.8016	0.8069	0.7994	0.7970	0.7842
SeedNE	**0.8155**	**0.8066**	**0.8229**	**0.8200**	**0.8413**	**0.8339**
ASeedNE	**0.8133**	**0.8034**	**0.8167**	**0.8107**	**0.8339**	**0.8234**

Table 5: Node Classification Result of Citeseer

Method	50%		70%		90%	
	Micro-F1	Macro-F1	Micro-F1	Macro-F1	Micro-F1	Macro-F1
DeepWalk	0.5782	0.5306	**0.6036**	**0.5426**	0.6235	0.5426
Node2Vec	0.5763	0.5094	0.5895	0.5237	0.6205	0.5346
GraRep	0.5522	0.4842	0.5503	0.4800	0.5813	0.4994
LINE	0.5534	0.5026	0.5674	0.5137	0.5904	0.5424
SeedNE	**0.5890**	**0.5419**	**0.5946**	**0.5463**	**0.6416**	**0.5795**
ASeedNE	**0.5914**	**0.5440**	0.5865	0.5354	**0.6295**	**0.5701**

it is less than 1, we have its gradient as follows:

$$\nabla l(u) = 2au \ . \tag{17}$$

Then, by maximizing Eq. (6) or Eq. (13), we can update u as follows:

$$u_{t+1} = u_t + 2\eta a u_t \ , \tag{18}$$

where $\eta > 0$ is the step size. Because $a > 0$ and u is initialized as 1, then the second term on the right-hand side is positive. As a result, u is increasing such that $l(u)$ is increasing. Then, we can gradually include difficult negative context nodes.

4.4 Results and Analysis

4.4.1 Node Classification. To evaluate the performance of network embedding, node classification is the most widely used method [4, 6, 21, 25]. In this paper, we also conduct node classification to show the performance of our proposed methods. The classifier used in this paper is Logistic Regression. Specifically, we use all nodes to train the network embedding method to get node representations. After that, we conduct node classification on these representations. To evaluate these network embedding methods comprehensively,

Table 6: Node Classification Result of Blogcatalog

Method	50%		70%		90%	
	Micro-F1	Macro-F1	Micro-F1	Macro-F1	Micro-F1	Macro-F1
DeepWalk	0.3927	0.2556	0.4047	0.2689	0.4169	0.2836
Node2Vec	0.3965	0.2582	0.4082	0.2698	0.4149	0.2828
GraRep	0.3674	0.2039	0.3830	0.2278	0.3869	0.2322
LINE	0.3518	0.1786	0.3597	0.1845	0.3597	0.1869
SeedNE	**0.4095**	**0.2646**	**0.4216**	**0.2843**	**0.4312**	**0.2884**
ASeedNE	**0.4106**	**0.2680**	**0.4243**	**0.2873**	**0.4326**	**0.3037**

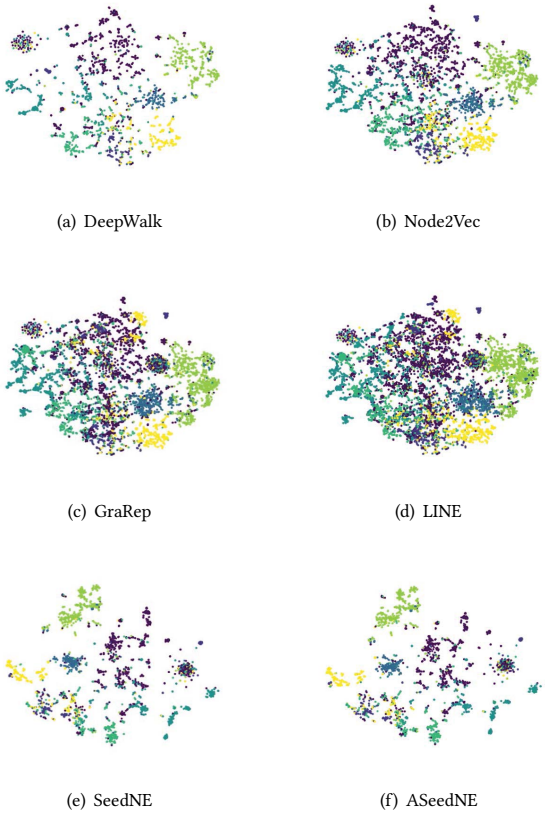
we randomly select {50%, 70%, 90%} nodes to train the classifier respectively. Then, the classifier's performance is evaluated on the rest nodes. To measure the classification performance, we employ Micro-F1 and Macro-F1 as metrics. The larger the two metrics are, the better the classification performance is.

The classification results are shown in Tables 2, 3, 4, 5, 6. In these tables, the best two results in each case are marked in bold. From these tables, we can find that both SeedNE and ASeedNE outperform the other state-of-the-art methods significantly. Specifically,

- In Table 2, the best two results are our proposed SeedNE and ASeedNE. Both of them have achieved significant improvement over the other baseline methods. The reason behind this improvement is that our methods can effectively discover informative negative context nodes according to the current embedding result, and then feed difficult negative context nodes gradually to train model parameters such that similar nodes are pushed together while dissimilar nodes are pushed away.
- In Table 2, comparing SeedNE with ASeedNE, we can find that ASeedNE can outperform SeedNE in most cases. The reason is that ASeedNE employs a generator to construct the negative sampling distribution, which has the larger capacity to capture the informativeness of each node than SeedNE. Thus, it can select more helpful negative context nodes to improve model's performance. Similar results can also be found in the other tables, which further verify the effectiveness of our proposed methods.

4.4.2 Network Visualization. Network visualization is another widely used method to evaluate the performance of network embedding algorithms. It is to project the learned node representations into a two-dimensional space such that we can visualize them. In this paper, we employ the well-developed tool t-SNE [17] to visualize node representations. Here, we only visualize the Cora dataset due to the space limitation. The visualization result is shown in Figure 1. From this figure, we have the following observations:

- In Figure 1(a), nodes scatter over the entire space such that these nodes have no a compact group structure, which means this method fails to push together similar nodes. Similar results can be found in Figure 1(b).
- In Figure 1(c), nodes from different classes are mixed with each other, especially for nodes in the center of this figure. This means that this method fails to push away dissimilar nodes. Figure 1(d) shows the similar result.



(a) DeepWalk (b) Node2Vec

(c) GraRep (d) LINE

(e) SeedNE (f) ASeedNE

Figure 1: The visualization of Cora dataset.

- In Figure 1(e), we can see that these nodes have compact group structures and the margin between different groups is large compared with baseline methods, which means our proposed SeedNE can push similar nodes together and push dissimilar nodes away. Thus, our method can achieve a better result on the node classification task. Our proposed ASeedNE has similar results, just as shown in Figure 1(f).

In conclusion, our proposed methods can successfully push similar nodes together and push dissimilar nodes away by sampling informative nodes in each iteration so that the embedding result is better than baseline methods.

Table 7: Link Prediction Accuracy

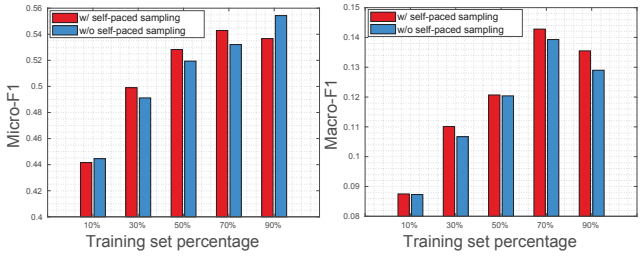
Method	Facebook	GR-QC
DeepWalk	0.9050	0.8354
Node2Vec	0.8900	0.7949
GraRep	0.9445	0.8899
LINE	0.9329	0.8847
SeedNE	**0.9532**	**0.9208**
ASeedNE	**0.9545**	**0.9230**

4.4.3 Link Prediction. In this section, we will evaluate the embedding result by using the link prediction task. The goal of this task is to predict the existence of an edge between two nodes. Specifically, Facebook and GR-QC are used in this experiment. We randomly select 10% edges from the original network as positive samples of the testing set. The remained network is used for learning node representation. Additionally, we randomly select the same number of edges from the undiscovered edges as negative samples of the testing set.

The link prediction accuracy of these two datasets is shown in Table 7. From this table, we can observe that our proposed methods outperform all the other baseline methods. Especially, the improvement of the GR-QC dataset is very significant. Specifically, the improvement over the best baseline method is around 4%, which further verifies the effectiveness of our proposed methods.

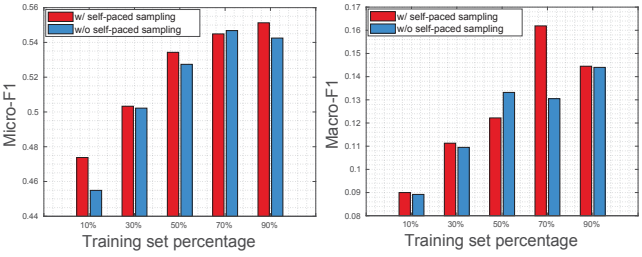
4.4.4 More Results. To further show the effect of self-paced sampling strategy, we compare our proposed methods with that without self-paced sampling strategy. In particular, this baseline method employs Eq. (5) to sampling negative context nodes at each iteration. Thus, it always feeds difficult negative nodes to train models. Here, we only show the result about the Wiki dataset due to the space limitation. Figure 2 shows node classification results about SeedNE. From this figure, we can find that our method can achieve better classification result for most cases based on both Micro-F1 and Macro-F1. Figure 3 shows node classification results about ASeedNE. Similarly, we can find that our method can beat that without self-paced sampling strategy for most cases, which further verifies the effectiveness of our proposed sampling strategy.

Moreover, in Figure 4, we show the threshold function value during training models. Here, we only show the result of Cora dataset due to the space limitation. Additionally, we only show the first 50 iterations because it is a constant value in the following iterations. Note that we let the parameter a change in the training process. Specifically, a is doubled in every 10 iterations. Intuitively, this operation will accelerate to include difficult negative context nodes. In practice, if a stays as a small value for a long time such that the threshold function value is small, only easy negative context nodes can be included so that the embedding result cannot be further refined. This is consistent with the cognitive activity of human beings. In Figure 4, the threshold function value increases slowly so that only easy negative context nodes are selected to train models at the beginning phase. With the training process going on, the threshold value becomes larger and larger. Therefore, more and more difficult negative context nodes are fed into the model, which is consistent with the intuition of our proposed methods.



(a) Micro-F1 (b) Macro-F1

Figure 2: Node classification result of Wiki dataset from the SeedNE method.



(a) Micro-F1 (b) Macro-F1

Figure 3: Node classification result of Wiki dataset from the ASeedNE method.

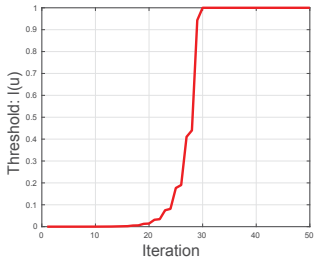


Figure 4: The threshold function value when training Cora.

5 CONCLUSION

In this paper, we proposed the novel self-paced network embedding methods. Our proposed methods can effectively capture the informativeness of each node. Based on the informativeness of nodes, we introduced a self-paced informativeness-aware sampling strategy. With this sampling strategy, our proposed methods can gradually select informative nodes to train model parameters. Extensive experiments on different data mining tasks have demonstrated the superior performance of our proposed methods.

REFERENCES

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. [n. d.]. TensorFlow: A System for Large-Scale Machine Learning.

[2] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. 2009. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*. ACM, 41–48.

[3] Bobby-Joe Breitkreutz, Chris Stark, Teresa Reguly, Lorrie Boucher, Ashton Breitkreutz, Michael Livstone, Rose Oughtred, Daniel H Lackner, Jürg Bähler, Valerie Wood, et al. 2008. The BioGRID interaction database: 2008 update. *Nucleic acids research* 36, suppl 1 (2008), D637–D640.

[4] Shaosheng Cao, Wei Lu, and Qiongkai Xu. 2015. Grarep: Learning graph representations with global structural information. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*. ACM, 891–900.

[5] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. In *Advances in neural information processing systems*. 2672–2680.

[6] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 855–864.

[7] Michael U Gutmann and Aapo Hyvärinen. 2012. Noise-contrastive estimation of unnormalized statistical models, with applications to natural image statistics. *Journal of Machine Learning Research* 13, Feb (2012), 307–361.

[8] Xiao Huang, Jundong Li, and Xia Hu. 2017. Accelerated attributed network embedding. In *Proceedings of the 2017 SIAM International Conference on Data Mining*. SIAM, 633–641.

[9] Lu Jiang, Deyu Meng, Shoou-I Yu, Zhenzhong Lan, Shiguang Shan, and Alexander Hauptmann. 2014. Self-paced learning with diversity. In *Advances in Neural Information Processing Systems*. 2078–2086.

[10] Lu Jiang, Deyu Meng, Qian Zhao, Shiguang Shan, and Alexander G Hauptmann. 2015. Self-paced curriculum learning. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*. AAAI Press, 2694–2700.

[11] Diederik Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[12] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).

[13] M Pawan Kumar, Benjamin Packer, and Daphne Koller. 2010. Self-paced learning for latent variable models. In *Advances in Neural Information Processing Systems*. 1189–1197.

[14] Jure Leskovec and Andrej Krevl. 2014. SNAP Datasets: Stanford Large Network Dataset Collection. http://snap.stanford.edu/data. (June 2014).

[15] Jundong Li, Harsh Dani, Xia Hu, Jiliang Tang, Yi Chang, and Huan Liu. 2017. Attributed network embedding for learning in a dynamic environment. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*. ACM, 387–396.

[16] Fan Ma, Deyu Meng, Qi Xie, Zina Li, and Xuanyi Dong. 2017. Self-paced co-training. In *International Conference on Machine Learning*. 2275–2284.

[17] Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-SNE. *Journal of machine learning research* 9, Nov (2008), 2579–2605.

[18] Matt Mahoney. 2009. Large text compression benchmark. *URL: http://www. mattmahoney. net/text/text. html* (2009).

[19] Andrew Kachites McCallum, Kamal Nigam, Jason Rennie, and Kristie Seymore. 2000. Automating the construction of internet portals with machine learning. *Information Retrieval* 3, 2 (2000), 127–163.

[20] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*. 3111–3119.

[21] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 701–710.

[22] Leonardo FR Ribeiro, Pedro HP Saverese, and Daniel R Figueiredo. 2017. struc2vec: Learning node representations from structural identity. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 385–394.

[23] John Schulman, Nicolas Heess, Theophane Weber, and Pieter Abbeel. 2015. Gradient estimation using stochastic computation graphs. In *Advances in Neural Information Processing Systems*. 3528–3536.

[24] James Steven Supancic III and Deva Ramanan. 2013. Self-paced learning for long-term tracking. In *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*. IEEE, 2379–2386.

[25] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. Line: Large-scale information network embedding. In *Proceedings of the 24th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 1067–1077.

[26] Lei Tang and Huan Liu. 2009. Relational learning via latent social dimensions. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 817–826.

[27] Daixin Wang, Peng Cui, and Wenwu Zhu. 2016. Structural deep network embedding. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 1225–1234.

[28] Jun Wang, Lantao Yu, Weinan Zhang, Yu Gong, Yinghui Xu, Benyou Wang, Peng Zhang, and Dell Zhang. 2017. Irgan: A minimax game for unifying generative and discriminative information retrieval models. In *Proceedings of the 40th International ACM SIGIR conference on Research and Development in Information Retrieval*. ACM, 515–524.

[29] Xiao Wang, Peng Cui, Jing Wang, Jian Pei, Wenwu Zhu, and Shiqiang Yang. 2017. Community Preserving Network Embedding. (2017).

[30] Cheng Yang, Zhiyuan Liu, Deli Zhao, Maosong Sun, and Edward Y Chang. [n. d.]. Network Representation Learning with Rich Text Information.

[31] Weinan Zhang, Tianqi Chen, Jun Wang, and Yong Yu. 2013. Optimizing top-n collaborative filtering via dynamic negative item sampling. In *Proceedings of the 36th international ACM SIGIR conference on Research and development in information retrieval*. ACM, 785–788.