PPPNE: Personalized Proximity Preserved Network Embedding

Ge Fan[1,4,*], Biao Geng[2,*], Jianrong Tao[3,5], Kai Wang[3], Changjie Fan[3], Wei Zeng[1]

[1]Center for Artificial Intelligence and Smart Health, University of Electronic Science and Technology of China, Chengdu, China
[2]College of Engineering, Carnegie Mellon University, Pittsburgh, USA
[3]Fuxi AI Lab, NetEase inc., Hangzhou, China
[4]Lgame, Tencent, Shengzhen, China
[5] Polytechnic Institute, Zhejiang University, Hangzhou, China

Abstract

After being proved extremely useful in many applications, the network embedding has played a critical role in the network analysis. Most of recent works usually model the network by minimizing the joint probability that the target node co-occurs with its neighboring nodes. These methods may fail to capture the personalized informativeness of each vertex. In this work, we propose a method named Personalized Proximity Preserved Network Embedding (PPPNE) to adaptively capture the personalization of vertices based on the personalized ranking loss. Our theoretical analysis shows that PPPNE generalizes prior work based on matrix factorization or neural network with single layer, and we argue that preserving personalized proximity is the key to learning more informative representations. Moreover, to better capture the network structure in multiple scales, we exploit the distance ordering of each vertex. Our method can be efficiently optimized with a vertex-anchored sampling strategy. The results of extensive experiments on five real-world networks demonstrate that our approach outperforms state-of-the-art network embedding methods with a considerable improvement on several common tasks including link prediction and vertex classification. Additionally, PPPNE is efficient and can be easily accelerated by parallel computing, which enables PPPNE to work on large scale networks.

Keywords: network embedding, learning to rank, personalized proximity preserving

1. Introduction

During the development of mobile internet and the advancement of decentralization technology, network information plays an essential role in knowledge discovery in databases. However, networks are unstructured and cannot be readily used as structured knowledge repositories which are indispensable in modern machine learning. To address this problem, network embedding approaches are proposed and widely adopted by many applications in internet systems, such as social recommendation and visualization [1].

The key of network embedding lies in finding a low-dimensional vector representation for each vertex to preserve the local and global structure information in the learned vector space [2]. Plenty of research efforts are devoted to preserving network structure by maximizing the proximities between vertices and their neighbors. The difference is that early works concentrate on first-order proximity while the recent incline to preserve high-order proximity or combine them. However, there are still two major challenges:

The personalization of vertices. The concept of personalization is initially introduced in the recommender system where individual preferences are diverse and items recommended for them should be distinctive [3]. However, few of research efforts concentrate on the personalization of vertices in the field of network embedding. For example, certain number of works minimize the joint probability of all links co-occurring in the whole network [4]. All vertices in the networks are treated equally by such method. Nevertheless, properties of vertices are diversified in the real networks. Some vertices have hundreds of neighbors while certain vertices have only a few neighbors. As a consequence, each vertex should be treated in a personalized way when the model is trained.

The natural ranking of vertices. Intuitively, the target vertex is closer to its 1-hop neighbors than its 2-hop neighbors and certain previous works consider the natural ranking of vertices. For instance, Bojchevski et al. [5] proposed the Graph2Gauss method which exploits the natural ranking of neighboring nodes to capture the network structure. However, the number of higher-order neighbors is normally much larger than the number of lower-order

*This work is done when Ge Fan and Biao Geng worked as intern at Fuxi AI Lab, NetEase.

Email addresses: fange@std.uestc.edu.cn (Ge Fan), bgeng@andrew.cmu.edu (Biao Geng), hztaojianrong@corp.netease.com (Jianrong Tao), wangkai02@corp.netease.com (Kai Wang), fanchangjie@corp.netease.com (Changjie Fan), zwei504@uestc.edu.cn (Wei Zeng)

neighbors. As a result, there exists bias in the procedure of instance sampling, whereas existing methods fail to balance such bias.

In order to address the above two challenges, we propose the personalized proximity preserved network embedding approach which borrows ideas from the Bayesian personalized ranking method [6], as illustrated in figure 1.

Personalization: We exploit a personalized ranking loss to capture the personalization of each vertex. More specifically, we treat first-order and high-order neighbors as the specific "context" for the target vertex and only the context nodes are used to learn representations of the target vertex (as depicted in the (c) stage in figure 1). It is similar to the scenarios in the recommender system where users' collected/purchased items are utilized to learn their features.

Natural ranking: In our mthod, the target vertex is closer to its lower-order neighbors than its higher-order neighbors, e.g. closer to its first-order neighbors than its second-order neighbors. Since a vertex usually has much more second-order neighbors than its first-order neighbors, we take advantage of the regularization method to balance such bias.

To summarize, the contributions of this work are listed as follows:

- We design an unsupervised personalized ranking formulation, preserving generalized first-order and second-order proximity jointly, to capture the personalized informativeness of vertices.
- We propose a novel model (PPPNE) for network embedding, leveraging the natural ranking of each vertex *w.r.t.* their neighborhoods via the proposed ranking formulation, which extends the capability of similarity measure in the complex networks.
- We perform experiments over five real-world networks and three frequently used applications , including link prediction, vertex classification and visualization. The results show that our methods can outperform all state-of-art baselines and achieve significant improvement (e.g. 11.57% relative improvement in AUC on link prediction task).

2. Related Work

It is a common way to represent a network in a sparse matrix form. However, as the amount of vertices of the network can be tremendous, the dimension curse is unavoidable. To address this problem, various network embedding methods have been developed. Among those methods, the earlier ones [7, 8] usually acquire a low dimension representation by decomposing the affinity matrix

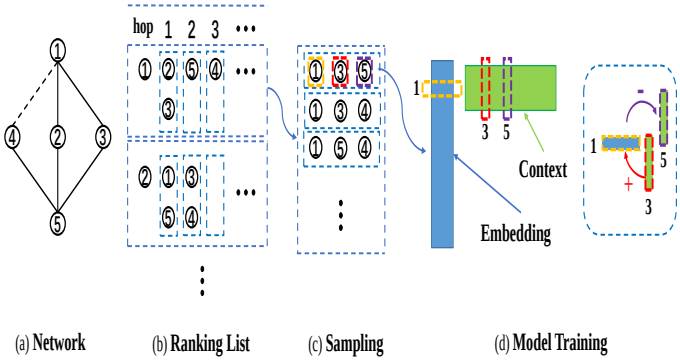


Figure 1: Framework of PPPNE. (a) An example network. (b) Searching q-hop neighborhoods for each vertex. (c) For each hop, sampling one vertex and acquiring all corresponding triplets. (d) Updating parameter for each sample. We make positive vertex (red) close to anchor vertex (yellow) and negative vertex (violet) away.

extracted from the network input. Locally Linear Embedding [7] aims to find embeddings which preserve distances within local neighborhoods and Spectral Cluster [8] uses the smallest eigenvectors of Laplacian matrix of the network as the representation, Graph Factorization (GF) [9] learns the representation by factorizing the normalized adjacency matrix. However, these conventional methods fail to take the high-order proximity into account and usually suffer from computational complexity caused by matrix decomposition.

Nowadays, new methods have been proposed for capturing the high-order proximity. DeepWalk [2] introduces the Skip-Gram model from word embedding by producing corpus with random walk technique. LINE [10] defines first-order and second-order proximity, taking a similar idea but refining the walk length as one to preserve local and global structural information in the large-scale network. Node2vec [11] extends the idea by adopting the potentially biased random walk to keep homophily and structural similarity. Since those random walk based methods are equivalent to factorizing high-order proximity matrices [4], there are many network embedding methods trying to preserving high-order proximity directly with matrix factorization technique [12, 13, 14, 15]. GraRep[12] considers different powers of the adjacency matrix to depict higher-order proximities and [13] preserves community structure of the network by non-negative matrix factorization. HOPE[14] algorithm incorporates high-order proximity measurements (e.g. Katz Index) into its framework for directed network. AROPE[15] extends it with a scalable eigen-decomposition solution to derive the embedding vectors and shift them between proximities of arbitrary order. Hierarchical Negative Sampling (HNS) [16] is proposed to draw more appropriate negative samples in network widely embedding methods by exploiting the rank N neighbor information in negative sample selection. Locality-aware meta-learning framework (meta-tail2vec)[17] can work with embedding models, such as

deepwalk and GraphSAGE, for refining tail vertex embeddings. This method personalize a regression model for each tail vertex, and learn it at different localities. [18] verifies that bagging methods for the homogeneous ensemble could be successfully applied to network embedding.

Deep neural networks [19, 20] are also used to capture the high-order proximity by seeking an effective non-linear mapping from original network to low-dimensional vector space. Auto-encoder[21, 19, 22] is a widely used architectural paradigms for learning latent representations. For instance, SDNE [21] uses auto-encoders to reconstruct the vertex's neighborhood, which is seen as the second-order proximity, and jointly preserves first-order proximity with Laplacian eigenmaps. DRNE [23] uses layer-normalized LSTM to aggregate representations of the vertex's neighborhood. Graph Diffusion Network (GDN) [24] replaces virgin aotu-encoder with Graph Diffusion Auto-encoder to maintain global information, and can dynamically preserve local and global consistency of graph. DONE [25] learns low-dimensional representation based on semi-supervised stacked sparse auto-encoder. ANE [26] learn network representations by leveraging the principle of adversarial learning. Adversarially regularized graph auto-encoder (ARGA) [27] learn smoothly embeddings by adopting adversarially regularized autoencoders. Directed Graph embedding framework based on Generative Adversarial Network (DGGAN) [28] jointly learns each vertex's source and target vectors by a discriminator with two generators.

More recently, graph convolutional network (GCN) [29] have achieved impressive performance for processing network data. The fundamental ideal of GCN is that iteratively modeling the attributes from neighbor vertices to a target vertex by an aggregating neural network. Simplified graph convolution (SGC) [30] speedup GCN through removing nonlinearities and collapsing weight matrices between consecutive layers. Adaptive multi-channel graph convolutional networks (AM-GCN) [31] extracts the specific and common embeddings from multi-channel sources, including the feature of vertices, topological structures, and their combinations simultaneously, and learns adaptive importance weights of the embeddings via a attention mechanism. Position-aware Graph Neural Networks (P-GNNs) [32] is proposed for computing position-aware vertex embeddings. Despite their success, most GCN based methods ignore the personalization of vertices since the aggregation function is defined as the summarization or average of neighboring feature.

Although personal ranking model [6] has been widely applied in information retrieval such as recommender systems and search engines [6, 33], there is few works to exploit unsupervised personalized ranking in network embedding. To the best of our knowledge, most of existing methods [34, 5] are mainly designed for attributed networks.

3. Problem Definition

We follow [10] to define the network as follow:

Definition 1 (Network). A Network is defined as $G = (V, E)$, where $V = \{v_1, v_2, ..., v_n\}$ is the vertex set and $E = \{e_{i,j}\}_{i,j=1}^{n}$ is the edge set, each edge $e_{i,j}$ denotes the relationship between two vertices and is associated with a weight $s_{i,j} \geq 0$ where $s_{i,j} = 0$ denotes v_i and v_j not linked by an edge. For other linked edges, if E is unweighted, $s_{i,j} = 1$, if E is weighted, $s_{i,j} > 0$. If G is undirected, $s_{i,j} \equiv s_{j,i}$, if G is directed, $s_{i,j} \not\equiv s_{j,i}$.

Note that while truly negative edge (e.g. review networks [35]) weights are possible , in this study, those unlinked edges, $s_{i,j} = 0$, are called negative edges. In a network, the neighbourhood of vertex v_i is the set of all vertices adjacent to v_i, formulated as $N(v_i) := \{v_j | v_j \in V, s_{i,j} > 0\}$.

Preserving network structures is one of the most important requirements for conducting the embedding. Network structures are usually implied by proximities of vertices which are divided as first-order proximity and second-order proximity defined in [10]. Recently, preserving high-order proximity by higher-hop neighbor relationship shows improved performance [14, 15]. In order to unify them, we define the generalized first-order proximity and generalized second-order proximity as below:

Definition 2 (Generalized First-order Proximity). The generalized first-order proximity in a network describes the immediate pairwise proximity between two vertices. For any vertices pair, (v_i, v_j) is only measured by the similarity function f_{sim} with the embedding pair (z_i, z_j).

Definition 3 (Generalized Second-order Proximity). The generalized second-order proximity between a pair of vertices (v_i, v_j) in a network describes the mediate proximity between two vertices. Mathematically, let $p_{v_i} = (w_{v_i,1}, \ldots, w_{v_i,|V|})$ denote the generalized first-order proximity of v_i with all the other vertices, then the generalized second-order proximity between v_i and v_j is determined by the similarity between p_{v_i} and p_{v_j}.

Note that the first-order (second-order) proximity in prior works [10, 21] can be seen as a special case of our generalized first-order (second-order) proximity by only considering 1-hop neighborhoods. In this paper, with no special mention, we represent generalized first-order (second-order) proximity as first-order (second-order) proximity for simplicity.

4. Proposed Methodology

4.1. Preserving Proximity by Personalized Ranking

For a vertex set $\{v_i, v_j, v_k\}$, if v_j is considered as closer to v_i than v_k (i.e. v_i, v_j are linked but v_i, v_k aren't), we mark

the set containing all satisfying triplets as D_s. Following [6], we can derive our personalized ranking loss as:

$$\mathcal{L} = -\sum_{(v_i,v_j,v_k)\in D_s} \log(\sigma(z_{i,j,k}(\Theta))) + \lambda_\Theta \|\Theta\|^2 \quad (1)$$

where $z_{i,j,k}(\Theta)$ is an arbitrary function of the model parameter Θ which captures the ranking relationship between $\{v_i, v_j, v_k\}$, σ is sigmoid function and λ_Θ is the hyper-parameter controlling the regularization term.

4.1.1. First-order Proximity Preserving

Inner product is a popular measurement to preserve first-order proximity and has been probed extensively in literature. In order to transform GF [9], the basic form for using inner product, we define $z_{i,j,k}(\Theta) = z_i^T z_j - z_i^T z_k$ and write the first-order proximity preserved loss as:

$$\mathcal{L}_{1st} = -\sum_{(v_i,v_j,v_k)\in D_s} \log(\sigma(z_i^T z_j - z_i^T z_k)) + \lambda \|\mathbf{z}\|^2 \quad (2)$$

where z_i, z_j and z_k denote the embedding vectors of v_i, v_j and v_k. λ denotes the regularization for matrix z formed by z_i. Note that though our Eq. 2 adopts the simplest form to model the first-order proximity, some recent works are easily proved as the equivalence of Eq. 2 under our framework if model function is monotonically increasing with the result of inner product.

4.1.2. Second-order Proximity Preserving

The second-order proximity assumes that vertices sharing many connections to other vertices are similar to each other. In order to utilize this kind of proximity, a common way is to treat each vertex as a specific "context" and vertices with similar distributions over the "contexts" are assumed to be similar. We follow it assuming each vertex plays two roles: embedding vector itself and a specific "context" for others. Replace $z_{i,j,k}(\Theta)$ with $z_i^T z'_j - z_i^T z'_k$ and loss function is defined as follow:

$$\mathcal{L}_{2nd} = -\sum_{(v_i,v_j,v_k)\in D_s} \log(\sigma(z_i^T z'_j - z_i^T z'_k)) + \lambda \|\mathbf{z}\|^2 + \lambda' \|\mathbf{z}'\|^2 \quad (3)$$

where z'_j (z'_k) denotes the representation of v_j (v_k) when it is treated as a specific "context", and λ' denotes the regularization for "context" vectors.

Similarly, since prior works [36, 10, 4] show that some recent methods (i.e. LINE_{2nd}) can be unified as matrix factorization methods, Eq. 3 generalizes aforementioned models based on matrix factorization or neural network with single layer.

4.1.3. Combined Preserving

So far we have developed two methods preserving the first and second order proximity respectively. Then there are different ways to combine two models under our framework. Concatenating the embeddings trained by the two methods for each vertex is a simple one but may re-weight difficultly in unsupervised tasks [10]. In this paper, we jointly train the Eq. 2 and 3 as follow:

$$\begin{aligned}\mathcal{L} = &-\sum_{(v_i,v_j,v_k)\in D_s} \log(\sigma(z_i^T z'_j - z_i^T z'_k)) \\ &+ \alpha \|\mathbf{z} - \mathbf{z}'\|^2 + \lambda \|\mathbf{z}\|^2 + \lambda' \|\mathbf{z}'\|^2\end{aligned} \quad (4)$$

where α is the hyper-parameter controlling the effect of distance between content vector z and context vector z'. The regularization term $\|\mathbf{z}-\mathbf{z}'\|^2$ can be considered as a balance between first-order neighbors and second-order neighbors. It is obvious that formula 4 and 2 is equivalent when $\alpha = \infty$, and formula 4 and 3 is equivalent when $\alpha = 0$.

4.2. PPPNE Model

Firstly, we define the distance $ds(i, j)$ as the shortest length between vertex v_i and v_j.

Intuitively, those pairs of vertices having shorter distance are closer in the embedding space. Thus, the complete satisfying set can be formulated as:

$$D_s = \{(v_i, v_j, v_k) \in V^3 | ds(v_i, v_j) < ds(v_i, v_k)\} \quad (5)$$

We can optimize Eq. 4 with stochastic gradient descent (SGD). But it is expensive to calculate all the combinations of all samples especially when the network is large. To address the problem and balance the updating between short-distance pairs and long-distance pairs, we propose a vertex-anchored sampling strategy presented in Algorithm 1 in each iteration.

Algorithm 1 Vertex-anchored sampling strategy

Input:
  Q: maximum distance;
Output:
  D'_s: sampling set of D_s;

```
set V_s = {} ;
for v_i ∈ V do
    for q from 1 to Q − 1 do
        randomly select v_j ∈ V into V_s where ds(i, j) = q;
    end for
    randomly select v_j ∈ V into V_s where ds(i, j) ≥ Q;
end for
D'_s = {(v_i, v_j, v_k) ∈ V_s^3 | ds(v_i, v_j) < ds(v_i, v_k)};
```

The complexity of computing the original Eq. 4 is $O(N^3)$, where N is the number of vertices of a network. As shown in Fig. 1, q-hop neighbors of all vertices are calculated and cached in the beginning, when Algorithm 1 is adopted, the complexity is reduced to $O(QN)$. Since $Q \ll N$ and setting Q to a small value (i.e. 2) also shows comparable performance [5], the complexity can be considered as $O(N)$. We describe the detailed training process in Algorithm 2.

5. Experiments

5.1. Experiment Setting

5.1.1. Datasets

In order to comprehensively measure the performance of our proposed method, we choose 5 different real-world

Algorithm 2 Optimizing PPPNE

Input:
Iter: training iterations;
γ : learning rate;
$G = (E, V)$: Network;
Output:
Θ: all model parameters;
randomly initialize all parameters ;
for it from 1 to Iter do
  Sampling D'_s with Algorithm 1
  for each (v_i, v_j, v_k) in D'_s do
    set $\mathcal{L}$ use Eq. 4 with input (v_i, v_j, v_k);
    set $z_i \leftarrow z_i - \gamma \frac{\partial \mathcal{L}}{\partial z_i}$;
    set $z_j \leftarrow z_j - \gamma \frac{\partial \mathcal{L}}{\partial z_j}$;
    set $z_k \leftarrow z_k - \gamma \frac{\partial \mathcal{L}}{\partial z_k}$;
    set $z'_i \leftarrow z'_i - \gamma \frac{\partial \mathcal{L}}{\partial z'_i}$;
    set $z'_j \leftarrow z'_j - \gamma \frac{\partial \mathcal{L}}{\partial z'_j}$;
    set $z'_k \leftarrow z'_k - \gamma \frac{\partial \mathcal{L}}{\partial z'_k}$;
  end for
end for

datasets from different domains. Protein-Protein Interactions (PPI) is a subgraph of biological network for Homo Sapiens, whose edges represent the pairwise interactions between proteins. We choose the proteins as nodes which can be labelled by the hallmark gene sets [37]. BlogCatalog [1] is a social network drawn from BlogCatalog, whose edges denote the social relationship of the bloggers listed on the website. Each user is labelled by at least one category. MMORPG [2] is a social network on a Justice game. We use the friendship data between players provided by a server. Cora [3] is a citation network of research papers. Vertices represent the published papers and edges represent that if the paper cites or is cited by other papers. Each paper is associated with one categories. Gowalla [4] is a large location-based social network. The statistics of all networks are given in Table 1.

Table 1: Statistics of the benchmark networks.

Dataset	$\|V\|$	$\|E\|$	labels	Avg. Degree
Cora	2,708	5,278	7	3.90
PPI	2,909	27,363	50	18.81
BlogCatalog	10,312	333,983	39	64.78
MMORPG	46,578	220,881	–	9.48
Gowalla	196,591	950,327	–	9.67

[1] http://socialcomputing.asu.edu/pages/datasets
[2] https://n.163.com
[3] https://linqs.soe.ucsc.edu/data
[4] https://snap.stanford.edu/data/loc-gowalla.html

5.1.2. Baseline Methods

We compare our instances with several existing network embedding methods as follows:

- Spectral Clustering (SC) [8] [5] . It is a state-of-art variant of LE, which computes the first d eigenvectors of normalized Laplacian matrix. This method is used to benchmark the performance of NE methods which only use first-order proximity .
- Graph Factorization (GF) [9] [6] . It factorizes the normalized adjacency matrix. We follow [38] using SGD to minimize the objective function.
- LINE [10] [7] . It learns two separate embeddings LINE_{1st} and LINE_{2st} for preserving the first and second order proximity respectively, then concatenates those two parts into a long vector. We report the best performance from those three models.
- Node2vec [11] [8] . It generalizes DeepWalk by adopting potentially biased random walks. These random walks follow the breadth-first sample and depth-first sample strategies with the flexible hyperparameters p and q.
- Graph2Gauss (G2G_oh)[5] [9] . This model learns to embed each node as a low-dimensional Gaussian distribution by ranking similarity based on the shortest path between nodes. As all datasets are plain network, we follow the author using one-hot encoding of the nodes as attributes in the paper.
- AROPE [15] [10] . This is a state-of-the-art method based on SVD, which derives the embedding vectors and shifts them between proximities of orders.
- VERSE [39] [11] . This is a state-of-the-art embedding method. VERSE reconstructs the distribution of chosen similarity measure for each node.
- ACNE [40] [12] . This is an adversarial learning method for network embedding. ACNE jointly learns the vertex and community representations in GANs framework.
- Meta-tail2vec [17] [13] . This model refines the long tail vertex embedding with a personalized regression model, and uses a locality-aware meta-learning framework to alleviate overfitting problem.

[5] https://github.com/scikit-learn/scikit-learn
[6] https://github.com/palash1992/GEM
[7] https://github.com/carpedm20/LINE
[8] https://github.com/aditya-grover/node2vec
[9] https://github.com/abojchevski/graph2gauss
[10] https://github.com/ZW-ZHANG/AROPE
[11] https://github.com/xgfs/verse
[12] https://github.com/junyachen/ACNE
[13] https://github.com/smufang/meta-tail2vec

We exclude some other network embedding methods, such as GraRep [12] and M-NMF [13], for their scalability issues. We also exclude HOPE [14] since it can been seen as a special case of AROPE. For our methods, we present two variations:

- PPPNE. We propose this method to preserve the personalized proximity of network by optimizing Eq. 4. we adopt the mini-batch RMSProp [41] with a fixed learning rate of 0.001.
- PPPNE$_{1st}$. In order to study the effect of the ranking loss alone, we mark the variation of PPPNE which optimizes Eq. 2 as PPPNE$_{1st}$.

5.1.3. Parameter Settings

We implement our proposed method based on Tensorflow and randomly initialize model parameters with xavier initialization [42]. The hyper-parameters of proposed model and baselines are tuned by grid search on a small validation set, which we set as 10% in our experiments [15]. For Node2vec, we test both its in-out and return hyper-parameters in $\{0.25, 0.5, 1\}$. For LINE, we set the number of negative samples as 5. For AROPE, we test the order in $\{1, 2, 3, 4\}$. For VERSE, we set the $\alpha = 0.85$. For ACNE, we set the $\lambda = 10^{-5}$. For Meta-tail2vec, we choose DeepWalk as the base embedding model. Specifically, we set the dimension $d = 128$ for all methods.

5.2. Link Prediction

Link prediction, whose goal is to predict missing edges, is one of the most common applications in real world. In this section, we first hide a portion of the existing edges and learn the embeddings of vertices in the rest. We then predict the held-out edges by applying the similarity function of pairs between two vertices with the obtained embedding. With no special mention in this paper, we hide about 20% edges of origin network, which are seen as positive samples, and randomly select no-edge vertex pairs as negative samples with equal number. The Area Under ROC Curve (AUC) and Average Precision (AP) [5] are adopted as the evaluation metrics. Besides, we measure the relative improvement (RelaImpr) over baseline models. Since the value of AUC from a random guesser is 0.5, we follow [43] define the RelaImpr of AUC. We report the main results in Table 2, after analyzing the results, some observations are listed as follows:

- Our two proposed methods significantly outperform all baseline methods on five networks, especially for AP metric. In addition, even the PPPNE$_{1st}$ which only preserves first order proximity have better performance than baselines in all cases. It demonstrates the strong ability of our proposed methods to tackle the personalization of vertices and predict the unobserved edges.
- G2G_oh is another model using ranking loss similar to our model, but it achieves a poorer performance than both of our methods. As depicted above, the target node normally has more higher-order neighbors than lower-order neighbors and G2G_oh doesn't balance such bias in the process of instance sampling. In addition, G2G_oh employs K-L divergence to evaluate the similarity between vertices but paper [23] verified the K-L divergence may be not a proper similarity measure to capture the transitivity for the undirected networks.
- As a matter of fact, PPPNE$_{1st}$ has comparable results with the PPPNE, the situation of which is similar with LINE$_{1st}$ vs. LINE$_{2nd}$ in our experiments and prior works [44, 45]. One possible explanation is that PPPNE$_{1st}$ considers the first-order proximity, which is capable of capturing the local structure of the network. Both PPPNE$_{1st}$ and PPPNE are better than LINE since our methods preserve the personalization of vertices.

5.3. Vertex Classification

Vertex classification is another important task of network embedding. In order to measure the performance of above methods in vertex classification, following [21, 11], we first unsupervisedly embed each vertex as a low dimensional vector, which is usually seen as features of the classification application. Then we classify vertices with already existing labels. Specifically, we choose the one-vs-rest logistic regression provided by LIBLINEAR package [46] as our classifier. For all datasets, we randomly sample 10% to 90% of the vertices as the training samples and use the left vertices to test the performance. We use averaged Micro-F1 and Macro-F1 as metrics for evaluation [10]. We repeat this process 10 times and report the result in Table 3. From the result, we have following observations:

- Similar to link prediction task, our proposed methods achieve the best performance compared with other baseline methods in both metrics on all datasets. Specifically, PPPNE obtains 3.48% ~ 13.67% (average 8.91%) relative improvement with the best result of baseline methods in Macro-F1 and 3.22% ~ 12.22% (average 7.67%) relative improvement in Micro-F1. It demonstrates that our model can learn better representations than baselines in the classification task.
- Differing from link prediction task, PPPNE performs much better than PPPNE$_{1st}$, which verifies the necessity of considering second-order proximity in this unsupervised task.
- Compared with different sparsity of training data, the relative improvement of PPPNE over the baselines is more significant when the training percentage decreases. It is an important advantage for real-world applications since the labelled data is usually limited.

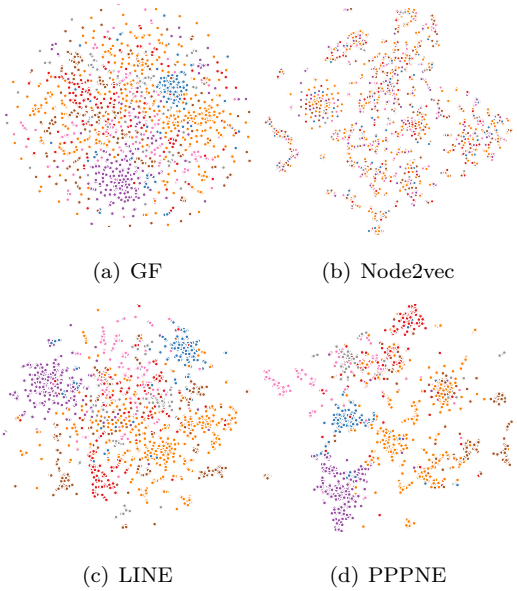
Table 2: The AUC and AP of link prediction. Bold values indicate the best results. Underlined values indicate the best results of all baseline Methods. *RelaImpr* shows the relative improvement between PPPNE with underlined values. The result shows that PPPNE outperforms all baseline in both AUC and AP.

	BlogCatalog		PPI		Cora		MMORPG		Gowalla	
Model	AUC	AP	AUC	AP	AUC	AP	AUC	AP	AUC	AP
SC	0.501	0.335	0.502	0.336	0.497	0.340	0.502	0.335	0.499	0.332
GF	0.591	0.381	0.605	0.396	0.681	0.622	0.608	0.461	0.717	0.570
$LINE_{1st}$	0.625	0.510	0.634	0.547	0.743	0.766	0.712	0.694	0.908	0.903
$LINE_{2nd}$	0.612	0.484	0.632	0.515	0.792	0.763	0.801	0.728	0.853	0.816
Node2vec	0.629	0.471	0.635	0.496	0.865	0.807	0.631	0.560	0.817	0.760
G2G_oh	0.833	0.817	0.784	0.777	0.851	0.876	0.803	0.804	0.945	0.950
AROPE	0.842	0.729	0.767	0.643	0.764	0.731	0.767	0.665	0.855	0.800
VERSE	0.737	0.550	0.740	0.557	0.839	0.794	0.770	0.688	0.925	0.891
ACNE	0.745	0.679	0.768	0.573	0.841	0.811	0.739	0.838	0.905	0.894
Meta-tail2vec	0.634	0.447	0.635	0.479	0.862	0.822	0.651	0.557	0.843	0.770
$PPPNE_{1st}$	0.880	0.874	0.822	0.834	0.913	0.929	0.818	0.839	0.959	0.969
PPPNE	0.886	0.876	0.827	0.841	0.925	0.939	0.825	0.852	0.963	0.970
RelaImpr	12.87%	7.22%	15.14%	8.24%	16.44%	7.19%	7.26%	5.97%	4.04%	2.11%

5.4. Visualization

Visualization is another critical application of network embedding. We follow a common experimental setting of previous works [5] to visualize the learned representations of Cora citation network. Firstly, we embed each node as a 128 dimensional vector and map those vectors into the 2-D space with t-SNE. Then we visualize each vector in Figure 2. We use different colors on the corresponded points which are labelled as different categories. Besides the visualization figure, the Kullback-Leibler divergence is a quantitative evaluation metric , and the lower K-L divergence indicates the better performance. The result is shown in Table 4.

From Figure 2, we can see that GF is not satisfactory since all points belonging to different categories are mixed with each other. For LINE and Node2vec, the clusters of different categories are formed but some parts of points are still mixed. For PPPNE, most of categories can be intuitively distinguished and the similar nodes are closer to each other than dissimilar nodes in the low-dimensional space. Table 4 reports the K-L divergence of each methods, from the result we can see PPPNE significantly outperforms all baseline methods, which also quantitatively demonstrates the superiority of our method.



(a) GF (b) Node2vec

(c) LINE (d) PPPNE

Figure 2: 2-D visualization on the Cora dataset. Each point indicates one research paper. Color of a point indicates the category of the paper, including Case Based, Genetic Algorithms, Neural Networks, Probabilistic Methods, Reinforcement Learning, Rule Learning and Theory.

5.5. Sampling Strategy

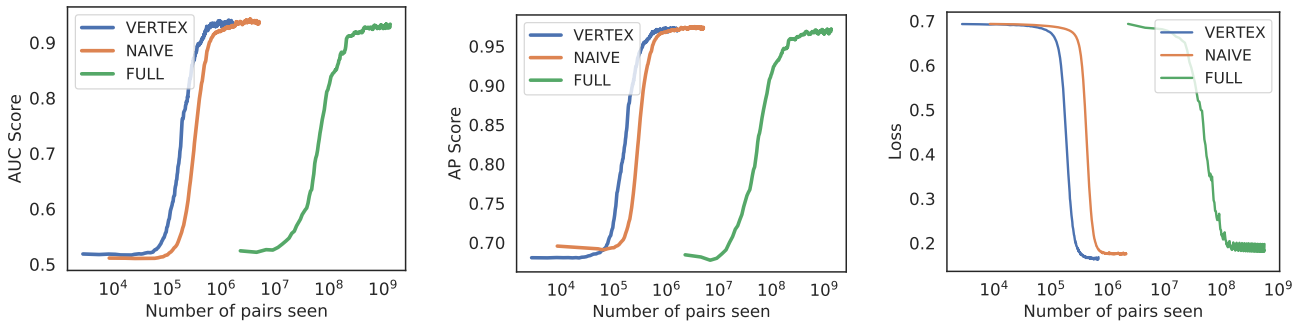
In order to investigate the effect of our vertex-anchored sampling strategy, we conduct several experiments on Cora network. We compared the difference between Full loss, naive edge-based sampling [10] and our vertex-anchored sampling strategy. Figure 3(a) and 3(b) show the AUC and AP score in the validation set for the link prediction task *w.r.t.* the number of triplet pairs. It reviews that both sampling strategies need about 3.72% number of pairs to acquire the identical performance as the full loss. Moreover, vertex-anchored sampling strategy converges faster than the naive edge-anchored sampling. Figure 3(c) shows that the vertex-anchored sampling strategy achieves significantly lower loss.

5.6. Efficiency Analysis

To evaluate the efficiency of PPPNE, we report the running time in Figure 4. All experiments are conducted in

Table 3: Performance of vertex classification *w.r.t* percent of training size. All results are expressed in percentage terms.

Algorithm	Macro F1									Micro F1								
	10%	20%	30%	40%	50%	60%	70%	80%	90%	10%	20%	30%	40%	50%	60%	70%	80%	90%
PPI																		
SC	8.42	10.88	12.42	13.59	14.35	14.77	15.46	15.43	15.66	13.73	16.50	18.21	19.28	19.91	20.12	20.88	20.97	21.47
GF	4.68	6.26	7.21	8.20	8.94	9.39	9.76	9.93	10.31	10.72	12.74	14.01	15.10	15.78	16.16	16.42	16.78	17.12
$LINE_{1st}$	9.68	11.75	13.02	13.68	14.35	14.84	15.01	14.78	14.02	14.41	16.17	17.41	17.93	18.58	19.21	19.26	19.17	18.79
LINE	11.91	14.12	15.21	16.07	16.54	16.65	17.14	17.48	17.03	15.66	17.24	18.27	19.03	19.60	19.90	20.48	20.76	21.00
Node2vec	12.18	14.11	15.57	16.37	17.05	17.58	17.79	17.81	18.00	15.41	17.08	18.28	19.02	19.84	20.42	20.86	21.18	21.67
G2G_oh	9.35	11.28	12.36	13.17	13.52	13.90	14.44	14.02	14.38	15.37	17.29	18.41	19.43	19.82	20.23	20.80	20.49	20.94
AROPE	10.78	12.52	13.50	14.18	14.38	14.96	15.06	14.96	14.93	13.95	15.24	16.05	16.62	17.01	17.57	17.78	17.74	18.04
VERSE	10.31	11.77	12.85	13.73	14.16	15.06	15.40	15.90	15.72	13.04	14.02	15.08	15.99	16.57	17.48	18.05	18.66	19.18
ACNE	11.52	13.85	14.95	15.65	16.36	16.87	17.11	16.93	16.77	14.96	16.74	17.83	18.56	19.33	19.84	20.29	20.63	20.88
$PPPNE_{1st}$	12.19	14.66	16.06	16.81	17.39	17.63	18.17	18.06	18.08	16.76	18.82	20.00	20.74	21.33	21.61	22.13	22.18	22.34
PPPNE	13.70	16.05	17.46	18.57	19.12	19.55	19.46	19.87	19.40	17.56	19.40	20.66	21.75	22.21	22.81	22.77	23.47	23.28
RelaImpr	12.48	13.67	12.14	13.44	12.14	11.21	9.39	11.57	7.78	12.13	12.20	12.22	11.94	11.55	11.70	9.05	10.81	7.43
BlogCatalog																		
SC	11.17	14.33	16.18	17.45	18.22	18.92	19.35	19.48	19.54	26.35	29.36	31.21	32.44	33.04	33.74	34.33	34.45	34.92
GF	6.00	7.07	7.91	8.55	8.94	9.30	9.79	10.00	10.17	21.89	23.78	25.08	25.97	26.26	26.71	27.13	27.42	27.60
$LINE_{1st}$	14.60	17.27	18.66	19.81	20.30	20.73	21.16	21.45	21.42	28.54	31.09	32.77	33.96	34.55	35.10	35.42	35.61	35.93
LINE	18.20	20.92	22.23	23.38	24.47	24.85	25.40	25.98	26.12	31.79	33.91	35.30	36.52	37.70	38.30	38.77	39.33	39.43
Node2vec	19.83	22.83	24.50	25.73	26.51	27.13	27.69	27.88	27.55	34.78	37.08	38.50	39.48	40.01	40.73	40.97	41.04	41.02
G2G_oh	8.61	9.72	10.41	10.97	11.23	11.53	11.64	11.80	12.05	25.60	26.86	27.54	28.19	28.27	28.55	28.59	28.74	28.91
AROPE	13.91	15.54	16.25	17.00	17.12	17.18	17.33	17.68	17.22	25.23	27.80	29.44	30.66	31.37	31.86	32.12	32.29	32.58
VERSE	16.78	20.10	21.65	22.85	23.51	23.87	24.18	24.23	24.49	30.19	33.40	35.13	36.43	36.86	37.37	37.62	37.78	38.17
ACNE	17.62	20.73	22.17	22.92	23.96	24.54	24.77	25.01	24.89	32.15	34.80	36.20	37.18	38.04	38.44	38.68	39.04	39.19
$PPPNE_{1st}$	18.55	21.54	22.90	23.83	24.49	24.97	25.39	25.51	25.27	36.99	39.16	40.26	41.01	41.31	41.66	41.95	42.04	42.18
PPPNE	21.91	24.88	26.35	27.16	28.18	28.40	28.86	28.85	29.02	37.04	39.25	40.28	41.09	41.63	42.04	42.32	42.36	42.72
RelaImpr	10.49	8.98	7.55	5.56	6.30	4.68	4.23	3.48	5.34	6.50	5.85	4.62	4.08	4.05	3.22	3.30	3.22	4.14



(a) AUC Score (b) AP Score (c) Loss

Figure 3: Convergence *w.r.t.* sampling strategy. The result shows the vertex-anchored sampling strategy converges faster and achieves lower loss.

Table 4: K-L divergence for the Cora dataset.

Algorithm	GF	Node2vec	$LINE_{2nd}$	PPPNE
K-L divergence	2.09	0.94	0.87	0.72

a single PC with one I7 CPU and 128G memory. The result of Figure 4 shows that PPPNE is efficient, boosting the efficiency by more than 3 times over the baselines on three large datasets. The efficiency of PPPNE proved by experiments lays the foundation for applying PPPNE to large-scale networks. In addition, we apply different number of CPU cores in training to observe the efficiency variance since the asynchronous algorithms are easily used in the training [47]. Figure 4 shows the running time *w.r.t.* the number of CPU cores, from the results we can find that the time cost decreases almost linearly with increasing number of CPU cores, which shows PPPNE is able to keep the high efficiency in the distributed environment.

5.7. Scalability

To verify the scalability of PPPNE, we conduct experiments on synthetic networks. We first generate random networks with different sizes by Erdos Renyi model [48], then apply our PPPNE model to those random networks and report the running time. We fix the number of edges (as ten million) or the number of vertices (as one hundred thousand) in the experiments. The result of running time *w.r.t.* the number of nodes is plotted in Figure 5.

In Figure 5, we empirically observe that the running time of proposed method grows linearly, verifying the scalability of our method.

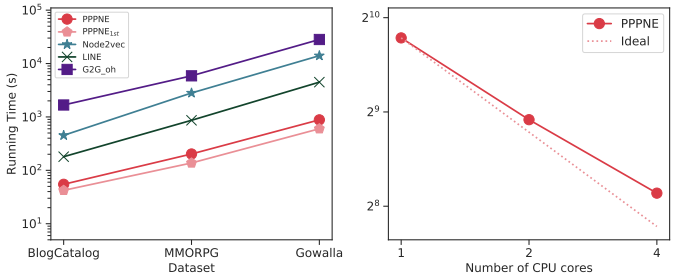


Figure 4: Efficiency of PPPNE. The result of left figure shows PPPNE can boost the efficiency by more than 3 times over state-of-the-art methods on BlogCatalog, MMORPG and Gewalla networks. The result of right figure shows the speed up of PPPNE is quite close to linear in the distributed environment.

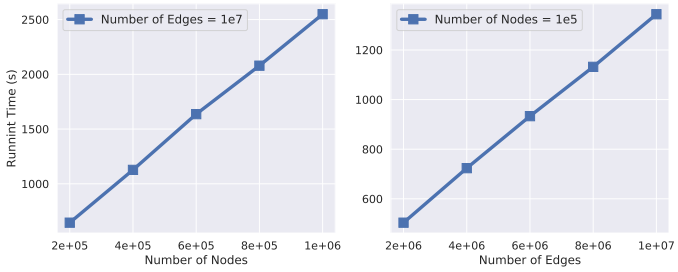


Figure 5: Scalability of PPPNE on Erdos Renyi Networks. The result shows that PPPNE has a linear time complexity *w.r.t.* the number of nodes and number of edges.

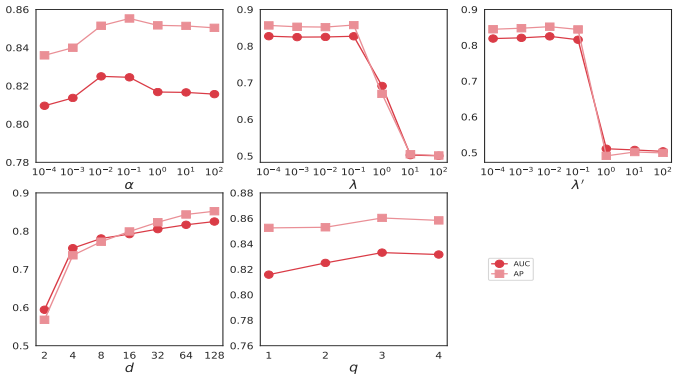


Figure 6: AUC and AP *w.r.t.* α, λ, λ', d and Q on MMORPG.

5.8. Parameter Sensitivity

In order to investigate how the hyper-parameters α, λ, λ', d and Q affect our model, we apply different values to observing the performance variance. The result in Figure 6 shows that our model is not considerably sensitive to the parameters. Specifically, for λ and λ', PPPNE is insensitive within a certain range (from 0.0001 to 0.1). λ and λ' are both typical L2 regularizations, using too large value may lead to under-fitting. Relatively, α does not limit the norm of embedding vectors and α is equivalent to the weight of Eq. 2 in Eq. 4. In consequence, a possible reason why α is more insensitive, is that those parameters learned by optimizing Eq. 4 are good enough to optimize both Eq. 2 and Eq. 3. For embedding size d, we test it in {2,4,8,16,32,64,128}. The result shows that PPPNE achieves a good performance even with small embedding size, which verifies that PPPNE is effective in tackling the complexity of network in the limit spaces. For neighbor hops Q, the result shows that when Q is small, our model can also learn a comparable embedding ($Q = 3$ achieve the best performance). One possible reason is that the sampling triples for a specific node are different in changed iterations, and thus as the number of iterations increases, even a simple sampling strategy can cover most of the positive edges, which captures most information of the network.

6. Conclusion

In this paper, we explored personalized ranking for network embedding. We devised a ranking formulation preserving first and second order proximity jointly, and proposed a inner product based method which captures more network structural information from the distance between vertices. Our theoretical analysis shows that PPPNE generalizes prior work based on matrix factorization or neural network with single layer. Moreover, our framework is simple and generic, and can be easily extended by modifying the function $z_{i,j,k}(\Theta)$ in Eq. 1. In addition, our method can be optimized by SGD with a vertex-anchored sampling strategy. We evaluate the performance of our method in several applications, including link prediction, vertex classification and visualization. Extensive experimental results on different real-world networks show that our proposed method outperforms the existing methods significantly in all tasks in both efficiency and effectiveness. In the future, we will study the embedding vectors from a dynamic network, which is more common in the real world. Dynamic network embedding methods should not only preserve the structural information, but also pay more attention to the development over time of the network. To build a dynamic network embedding method, we need to develop effective methods to learn the evolution patterns of a given network.

Acknowledgements

The work is supported by the National Natural Science Foundation of China (61502078, 61872062 and 61806045), and Fundamental Research Funds for the Central Universities (2672018ZYGX2018J050).

References

[1] P. Cui, X. Wang, J. Pei, W. Zhu, A survey on network embedding, TKDE 31 (5) (2019) 833–852. doi:10.1109/TKDE.2018.2849727.

[2] B. Perozzi, R. Al-Rfou, S. Skiena, Deepwalk: Online learning of social representations, in: SIGKDD, 2014, pp. 701–710.

[3] G. Adomavicius, A. Tuzhilin, Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions, TKDE 17 (6) (2005) 734–749.

[4] J. Qiu, Y. Dong, H. Ma, J. Li, K. Wang, J. Tang, Network embedding as matrix factorization: Unifying deepwalk, line, pte, and node2vec, in: WSDM, 2018, pp. 459–467. doi:10.1145/3159652.3159706.

[5] A. Bojchevski, S. Günnemann, Deep gaussian embedding of graphs: Unsupervised inductive learning via ranking, in: ICLR, 2018, pp. 1–13.

[6] S. Rendle, C. Freudenthaler, Z. Gantner, L. Schmidt-Thieme, Bpr: Bayesian personalized ranking from implicit feedback, in: UAI, 2009, pp. 452–461.

[7] S. T. Roweis, L. K. Saul, Nonlinear dimensionality reduction by locally linear embedding, science 290 (5500) (2000) 2323–2326.

[8] M. Belkin, P. Niyogi, Laplacian eigenmaps for dimensionality reduction and data representation, Neural computation 15 (6) (2003) 1373–1396.

[9] A. Ahmed, N. Shervashidze, S. Narayanamurthy, V. Josifovski, A. J. Smola, Distributed large-scale natural graph factorization, in: WWW, 2013, pp. 37–48.

[10] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, Q. Mei, Line: Large-scale information network embedding, in: WWW, 2015, pp. 1067–1077.

[11] A. Grover, J. Leskovec, node2vec: Scalable feature learning for networks, in: SIGKDD, 2016, pp. 855–864.

[12] S. Cao, W. Lu, Q. Xu, Grarep: Learning graph representations with global structural information, in: CIKM, 2015, pp. 891–900.

[13] X. Wang, P. Cui, J. Wang, J. Pei, W. Zhu, S. Yang, Community preserving network embedding., in: AAAI, 2017, pp. 203–209.

[14] M. Ou, P. Cui, J. Pei, Z. Zhang, W. Zhu, Asymmetric transitivity preserving graph embedding, in: SIGKDD, 2016, pp. 1105–1114.

[15] Z. Zhang, P. Cui, X. Wang, J. Pei, X. Yao, W. Zhu, Arbitrary-order proximity preserved network embedding, in: SIGKDD, 2018, pp. 2778–2786.

[16] J. Chen, Z. Gong, W. Wang, W. Liu, Hns: Hierarchical negative sampling for network representation learning, Information Sciences 542 (2021) 343–356.

[17] Z. Liu, W. Zhang, Y. Fang, X. Zhang, S. C. Hoi, Towards locality-aware meta-learning of tail node embeddings on networks, in: CIKM, 2020, p. 975–984. doi:10.1145/3340531.3411910.

[18] B. Zhang, J. Xiang, X. Wang, Network representation learning with ensemble methods, Neurocomputing 380 (2020) 141–149.

[19] S. Cao, W. Lu, Q. Xu, Deep neural networks for learning graph representations, in: AAAI, 2016, pp. 1145–1152.

[20] L. Heck, H. Huang, Deep learning of knowledge graph embeddings for semantic parsing of twitter dialogs, in: GlobalSIP, 2014, pp. 597–601.

[21] D. Wang, P. Cui, W. Zhu, Structural deep network embedding, in: SIGKDD, 2016, pp. 1225–1234. doi:10.1145/2939672.2939753.

[22] Y. Yang, H. Chen, J. Shao, Triplet enhanced autoencoder: model-free discriminative network embedding, in: IJCAI, AAAI Press, 2019, pp. 5363–5369.

[23] K. Tu, P. Cui, X. Wang, P. S. Yu, W. Zhu, Deep recursive network embedding with regular equivalence, in: SIGKDD, 2018, pp. 2357–2366.

[24] F. Li, Z. Zhu, X. Zhang, J. Cheng, Y. Zhao, Diffusion induced graph representation learning, Neurocomputing 360 (2019) 220–229.

[25] A. Fathy, K. Li, Done: Enhancing network embedding via greedy vertex domination, Neurocomputing 410 (2020) 71 – 82. doi:https://doi.org/10.1016/j.neucom.2020.05.055.

[26] Q. Dai, Q. Li, J. Tang, D. Wang, Adversarial network embedding, in: AAAI, 2018, pp. 2167–2174.

[27] S. Pan, R. Hu, S.-f. Fung, G. Long, J. Jiang, C. Zhang, Learning graph embedding with adversarial training methods, TCYB 50 (6) (2019) 2475–2487.

[28] S. Zhu, J. Li, H. Peng, S. Wang, P. S. Yu, L. He, Adversarial directed graph embedding, in: AAAI, 2021, pp. 1–8.

[29] T. N. Kipf, M. Welling, Semi-supervised classification with graph convolutional networks, in: ICLR, 2017, pp. 1–10.

[30] F. Wu, A. Souza, T. Zhang, C. Fifty, T. Yu, K. Weinberger, Simplifying graph convolutional networks, in: ICML, 2019, pp. 6861–6871.

[31] X. Wang, M. Zhu, D. Bo, P. Cui, C. Shi, J. Pei, Am-gcn: Adaptive multi-channel graph convolutional networks, in: SIGKDD, 2020, p. 1243–1253. doi:10.1145/3394486.3403177.

[32] J. You, R. Ying, J. Leskovec, Position-aware graph neural networks, in: ICML, 2019, pp. 7134—7143.

[33] F. Schroff, D. Kalenichenko, J. Philbin, Facenet: A unified embedding for face recognition and clustering, in: CVPR, 2015, pp. 815–823.

[34] S. Zhou, H. Yang, X. Wang, J. Bu, M. Ester, P. Yu, J. Zhang, C. Wang, Prre: Personalized relation ranking embedding for attributed networks, in: CIKM, 2018, pp. 823–832. doi:10.1145/3269206.3271741.

[35] S. Wang, J. Tang, C. Aggarwal, Y. Chang, H. Liu, Signed network embedding in social media, in: SDM, 2017, pp. 327–335.

[36] C. Yang, Z. Liu, D. Zhao, M. Sun, E. Y. Chang, Network representation learning with rich text information, in: IJCAI, 2015, pp. 2111–2117.

[37] A. Liberzon, A. Subramanian, R. Pinchback, H. Thorvaldsdóttir, P. Tamayo, J. P. Mesirov, Molecular signatures database (msigdb) 3.0, Bioinformatics 27 (12) (2011) 1739–1740.

[38] P. Goyal, E. Ferrara, Graph embedding techniques, applications, and performance: A survey, KBS 151 (2018) 78–94.

[39] A. Tsitsulin, D. Mottin, P. Karras, E. Müller, Verse: Versatile graph embeddings from similarity measures, in: WWW, 2018, pp. 539–548. doi:10.1145/3178876.3186120.

[40] J. Chen, Z. Gong, Q. Dai, C. Yuan, W. Liu, Adversarial learning for overlapping community detection and network embedding, in: ECAI, 2020, pp. 1–8.

[41] T. Tieleman, G. Hinton, Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude, COURSERA: Neural networks for machine learning 4 (2) (2012) 26–31.

[42] X. Glorot, Y. Bengio, Understanding the difficulty of training deep feedforward neural networks, in: AISTATS, 2010, pp. 249–256.

[43] G. Zhou, X. Zhu, C. Song, Y. Fan, H. Zhu, X. Ma, Y. Yan, J. Jin, H. Li, K. Gai, Deep interest network for click-through rate prediction, in: SIGKDD, New York, NY, USA, 2018, pp. 1059–1068.

[44] Z. Zhang, P. Cui, H. Li, X. Wang, W. Zhu, Billion-scale network embedding with iterative random projection, in: ICDM, 2018, pp. 787–796.

[45] Y. Lu, C. Shi, L. Hu, Z. Liu, Relation structure-aware heterogeneous information network embedding, in: AAAI, 2019, pp. 4456–4463.

[46] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, C.-J. Lin, Liblinear: A library for large linear classification, JMLR 9 (Aug) (2008) 1871–1874.

[47] F. Niu, B. Recht, C. Re, S. J. Wright, Hogwild!: A lock-free approach to parallelizing stochastic gradient descent, in: NIPS, 2011, pp. 693–701.

[48] L. Erdős, A. Knowles, H.-T. Yau, J. Yin, et al., Spectral statistics of erdős–rényi graphs i: local semicircle law, The Annals of Probability 41 (3B) (2013) 2279–2375.