Nonuniform Hyper-Network Embedding with Dual Mechanism

JIE HUANG and CHUAN CHEN, Sun Yat-sen University
FANGHUA YE, University College London
WEIBO HU and ZIBIN ZHENG, Sun Yat-sen University

Network embedding which aims to learn the low-dimensional representations for vertices in networks has been extensively studied in recent years. Although there are various models designed for networks with different properties and different structures for different tasks, most of them are only applied to normal networks which only contain pairwise relationships between vertices. In many realistic cases, relationships among objects are not pairwise and such relationships can be better modeled by a hyper-network in which each edge can connect an uncertain number of vertices. In this article, we focus on two properties of hyper-networks: nonuniform and dual property. In order to make full use of these two properties, we firstly propose a flexible model called Hyper2vec to learn the embeddings of hyper-networks by applying a biased second order random walk strategy to hyper-networks in the framework of Skip-gram. Then, we combine the features of hyperedges by considering the dual hyper-networks to build a further model called NHNE based on 1D convolutional neural networks, and train a tuplewise similarity function for the nonuniform relationships in hyper-networks. Extensive experiments demonstrate the significant effectiveness of our methods for hyper-network embedding.

CCS Concepts: • **Mathematics of computing** → **Hypergraphs**; • **Information systems** → Information systems applications;

Additional Key Words and Phrases: Network embedding, hyper-network, link prediction, dual network

ACM Reference format:
Jie Huang, Chuan Chen, Fanghua Ye, Weibo Hu, and Zibin Zheng. 2020. Nonuniform Hyper-Network Embedding with Dual Mechanism. *ACM Trans. Inf. Syst.* 38, 3, Article 28 (May 2020), 18 pages.
https://doi.org/10.1145/3388924

This work was supported by the National Key Research and Development Program (2016YFB1000101), the National Natural Science Foundation of China (61722214, 11801595), the Natural Science Foundation of Guangdong (2018A030310076), the Program for Science and Technology Planning Project of Guangdong Province of China (No. 2018B010109001), and the CCF Opening Project of Information System.
Authors' addresses: J. Huang, C. Chen (corresponding author), W. Hu, and Z. Zheng, School of Data and Computer Science, Sun Yat-sen University, Higher Education Mega Center, Panyu District, Guangzhou, China; emails: huangj285@mail2.sysu.edu.cn, chenchuan@mail.sysu.edu.cn; huwb7@mail2.sysu.edu.cn, zhzibin@mail.sysu.edu.cn; F. Ye, Department of Computer Science, University College London, Gower Street, London, UK; email: smartyfh@outlook.com.

1046-8188/2020/05-ART28 $15.00
https://doi.org/10.1145/3388924

1 INTRODUCTION

Networks play an important role in the real world; examples include the Internet, social networks, biological networks, and so forth. The pattern of connections and relationships between objects in a system can be represented as a network, where objects correspond to vertices and relations between objects correspond to the edges. Due to the rapid increase of network size and structural complexity, many graph learning methods are computationally infeasible for large-scale networks. To this end, a large number of network embedding methods [10, 14, 27, 33] have been proposed recently to obtain low-dimensional representations of vertices in the networks, meanwhile preserving the network structure as much as possible.

In the traditional definition of a network, each edge represents the connection or relationship between a particular pair of vertices. However, in many realistic situations, relationships among objects are not pairwise. For example, the number of authors in a paper may vary within a certain range, and the number of players in a multi-player game is also not fixed at pairwise. In these cases, it is better to introduce hyper-networks [6, 7, 40] to model such relationships among objects (Figure 1(a)).

Nevertheless, few network embedding methods can be applied to hyper-networks directly. A conventional way to learn for hyper-networks is to convert them into normal networks. For instance, clique expansion [32] converts each hyperedge to a set of normal edges (Figure 1(b)), and star expansion [1] transforms a hyper-network into a bipartite graph where each hyperedge is represented by an instance vertex which links to the original vertices it contains (Figure 1(c)). However, these transformations cause information loss of the original hyper-network either explicitly or implicitly. For instance, clique expansion converts all hyperedges equally to normal edges, which changes the relationships between vertices and loses the information of hyperedges since all possible edges implied by tuplewise relationships are treated equally and tuplewise relationships no longer exist. Until now, only several embedding methods for hyper-networks have been proposed [3, 36, 45, 49], and these models are mainly designed for hyper-networks with specific properties such as uniformity and indecomposability and thus cannot improve performance on a wide range of hyper-networks.

Different from the existing methods, in this work, our observations on hyper-networks are as follows:

- **Nonuniform:** The hyperedges in a hyper-network are usually nonuniform. For instance, the number of authors of an article varies in a certain amount. Nonuniform hyper-networks are much harder to deal with since a function with variable-length input is needed for predicting hyperedges which contain different numbers of vertices. In fact, most existing hyper-network embedding models are only suitable for uniform hyper-networks.
- **Dual property:** Different from normal networks, some hyper-networks have a special property that their dual networks have significant meanings [6]. The dual of a hyper-network can be obtained by swapping the roles of hyperedges and vertices. For instance, for a co-author hyper-network (Figure 1(a)), the dual is an article hyper-network where articles are vertices and the relationships between articles are the hyperedges. With such specific meaning, it's reasonable for us to take advantage of the dual hyper-networks to capture some important features of hyperedges such as the high-order proximities and structural properties of hyperedges which may be lost during modeling in the original hyper-networks.

Based on the factors described above, firstly, we propose a random walk–based model named **Hyper2vec** (previously presented in [18]). The Hyper2vec model generates biased second-order

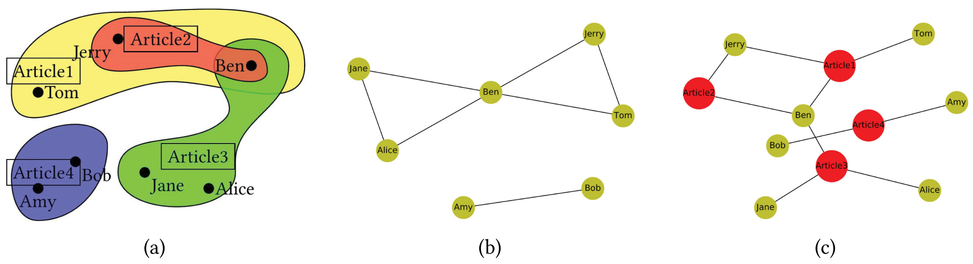


Fig. 1. Example illustrating co-author hyper-network. In this hyper-network, authors are vertices and articles are hyperedges. Tom, Jerry, and Ben cowrite Article1. Jerry and Ben cowrite Article2. Ben, Jane, and Alice cowrite Article3. Amy and Bob cowrite Article4. (a) Original hyper-network. (b) The clique expansion. (c) The star expansion.

random walks in hyper-networks and then put the random walks into the Skip-gram model to learn the embeddings of vertices. In addition, in order to combine the features of dual hyper-networks and deal with the nonuniform property, we design a 1D Convolutional Neural Network (1D-CNN) -based model called **NHNE** to train a tuplewise similarity function and update the embeddings to contain more features about the tuplewise relationships among vertices.

To evaluate the performance of the proposed model, we consider two widely considered prediction tasks for hyper-networks: link prediction [23], which predicts the existence of a hyperedge given a set of vertices, and vertex classification (or multi-label classification), which classifies each vertex with one (or more) label(s). Then we compare the performance of our model with several state-of-the-art network embedding algorithms. Experimental results demonstrate that Hyper2vec performs better in most cases, and with the combination of dual hyper-networks and tuplewise similarity function, NHNE has a further improvement.

In summary, our main contributions are as follows:

—We propose a flexible model for hyper-network embedding based on biased second-order random walks and a 1D-CNN model.
—We introduce a novel idea for combining the hyperedge features by taking advantage of the dual hyper-networks for hyper-network embedding and train a tuplewise similarity function to model the nonuniform relationships for link prediction.
—We conduct link prediction and vertex classification experiments on two real hyper-networks. The results demonstrate the effectiveness of the proposed model.

The remainder of this article is structured as follows. In Section 2, we review the related work. Section 3 gives notations and definitions of hyper-network. We introduce our model in detail in Section 4. In Section 5, we empirically evaluate our model on link prediction and classification tasks. In Section 6, we conclude and highlight some promising directions for future work.

2 RELATED WORK

Classical network embedding methods such as LLE [29], Laplacian eigenmaps [5], and IsoMap [35] suffer from both computational and statistical performance drawbacks. Recently, inspired by the Skip-gram model [24], researchers establish an analogy for networks by representing the vertices of a network as the words of a document. To process contexts, random walk models [41] are introduced to generate random traces over a network. Some representative methods include DeepWalk [27] and Node2vec [14]. DeepWalk adopts a first-order random walk model which uses a uniform transition probability to generate walks by treating walks as the equivalent of sentences. To make the random walk strategy more feasible, Node2vec uses a second-order random walk model to learn better representations by introducing two tunable parameters to control the tendency of random walks.

In addition, LINE [33] is proposed for large-scale network embedding, which can preserve the first- and second-order proximities. SDNE [37], SDAE [8], and SiNE [39] utilize deep neural networks to learn the embeddings of vertices. These studies make deep models fit network data and impose network structure and property-level constraints on deep models. In addition, it should be mentioned that there is an increase of interest in learning representations of vertices by neighborhood aggregation encoders [19, 30]; these models rely on the features and attributes of vertices and are hard applied to unsupervised tasks.

However, all the above methods only consider the pairwise relationships between data objects, thus they cannot be directly applied to hyper-networks. Some methods based on spectral clustering techniques are proposed to deal with hyper-networks [26, 42, 48], while the performance of embedding tasks is not satisfactory in most cases, because spectral clustering assumes that graph cuts are useful for classification [34]. And such assumptions are unsatisfactory in effectively generalizing across diverse networks. Heterogeneous Hypergraph Embedding (HHE) [49], Hypergraph Embedding (HGE) [45], Deep Hyper-Network Embedding (DHNE) [36], and Heterogeneous Hyper-Network Embedding (HHNE) [3] are recent models to learn the embeddings of hyper-networks. But all these models cannot learn the embeddings of a nonuniform hyper-network well. HHE [49] learns embeddings by solving eigenvector problems related to hyper-network Laplacian matrices, but it's both time consuming and space consuming. HGE [45] learns embeddings by incorporating multi-way relations into an optimization problem related to geometric mean and arithmetic mean, but it's not flexible and cannot capture the features of nonuniform hyper-networks well. DHNE [36] is a neural network–based model for hyper-network embedding, but it only applies to uniform heterogeneous hyper-networks with high indecomposable hyperedges. HHNE [3] utilizes a graph convolutional network to learn the embeddings of vertices, but it relies on the features of vertices and is difficult to apply to classification tasks in an unsupervised setting.

There are also some related works on bipartite network which can be applied to hyper-networks with the star expansion. Some models are designed on a user-item network for recommendation [15, 28]. In addition, Metapath2vec [11], HNE [9], EOE [44], HERec [31], and Shine [38] are embedding models for heterogeneous networks, which can also be applied to bipartite networks, but without good pertinence, these models cannot capture the information of the bipartite networks well, and thus cannot capture the information of the hyper-networks well. In addition, Bipartite Network Embedding (BiNE) [13] is a recent network embedding model designed for bipartite networks by biased random walks to preserve the long-tail distribution of vertices.

3 NOTATIONS AND DEFINITIONS

A *hyper-network* is represented as a hypergraph $G(V, E, w)$ with the set of vertices V, the set of hyperedges E, and the weight mapping function $w(e)$ associated with each hyperedge $e \in E$. A hyperedge e can be considered as a set of vertices. A hypergraph is called uniform if all hyperedges contain the same number of vertices, otherwise it is called nonuniform. A hypergraph is called heterogeneous if there are different kinds of vertices, otherwise it is called homogeneous. Given a set S, $|S|$ denotes the cardinality of S, so we have $n = |V|$ being the number of vertices and $m = |E|$ being the number of hyperedges. The *neighbors* of a vertex v is a vertex set $N_G(v) = \{x | \exists e, v \in e, x \in e\}$. If $v \in e$, we say that e is *incident* with v, and $E_G(v)$ is the set of hyperedges incident with

v. The *incident matrix* of a hypergraph is a matrix $H \in \mathbb{R}^{n \times m}$ with entries defined as follows:

$$h(v,e) = \begin{cases} 1, & \text{if } v \in e \\ 0, & \text{if } v \notin e \end{cases}. \tag{1}$$

The *dual* of a hypergraph $G(V,E)$ is a hypergraph $G^*(V',E')$ whose vertices correspond to the hyperedges of G, and whose hyperedge set is $\{E_G(v)|v \in V\}$. The incident matrix of G^* is the transpose of the incident matrix of G.

The degree of a vertex $v \in V$ and a hyperedge $e \in E$ are defined, respectively, as follows:

$$d(v) = \sum_{e \in E} w(e)h(v,e), \tag{2}$$

$$\delta(e) = |e| = \sum_{v \in V} h(v,e). \tag{3}$$

A *hypertrace* is a trace starting from v_s and ending at v_k, which is defined as a sequence of vertices and hyperedges $\{v_s, e_s, v_{s+1}, e_{s+1}, \ldots, e_{k-1}, v_k\}$ where the vertices may be repeated and $\{v_i, v_{i+1}\} \subseteq e_i$ for $s \le i \le k-1$.

4 METHODS

Random walk–based models are widely used in network embedding problems. In our work, we firstly generate random walks in hyper-networks to capture the high-order proximity of vertices. In particular, we establish a basic probabilistic model for hyper-network, and then use a second-order random walk to smoothly interpolate between Depth-First Search (DFS) and Breadth-First Search (BFS). After that, we introduce a degree biased random walk mechanism to correct the negative bias or introduce a positive bias for random walks. Secondly, we introduce the dual hyper-network mining strategy and design a neural network model to combine the features of both original hyper-network and dual hyper-network, meanwhile training a tuplewise similarity function to evaluate the nonuniform relationships in hyper-networks.

4.1 The First-Order Random Walk

The key to generate random walks in networks is to define the transition probability of one vertex to its neighbors. Traditional network embedding methods like DeepWalk adopt a uniform random walk strategy in normal networks. For hyper-networks, we introduce the transition strategy described in [48]. For a current vertex v, we firstly select a hyperedge e incident with v randomly according to the weight of e, and then select a vertex $x \in e$ as the next vertex of the random walk. Starting from a vertex and repeating the above process, we will end up with a hypertrace, which is the goal of random walks. In conclusion, the unnormalized transition probability of the first-order model is calculated as follows:

$$\pi_1(x|v) = \sum_{e \in E} w(e) \frac{h(v,e)}{d(v)} \frac{h(x,e)}{\delta(e)}. \tag{4}$$

4.2 The Second-Order Random Walk

To balance the homophily and structural equivalence [17] when conducting network embedding, it is effective to adopt a second-order random walk strategy [14]. In the second-order random walk, a random walker moves to the next vertex x based on not only the current vertex v but also the previous vertex u. To achieve this goal, we introduce a second-order random walk strategy in a hyper-network by using two parameters p and q. The neighbors of vertex v are divided into three

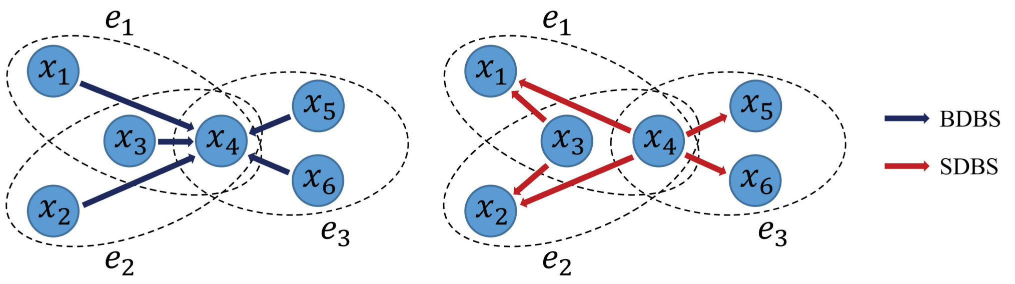


Fig. 2. Illustration of BDBS and SDBS in hyper-networks. In this simple case, BDBS guides random walks to the (local) centers, and SDBS guides random walks to the boundary.

categories: the previous vertex u, the neighbors of u, and the others. And we define the second-order search bias $\alpha(\cdot)$ as follows:

$$\alpha(x|v,u) = \begin{cases} \frac{1}{p}, & \text{if } x = u \\ 1, & \text{if } \exists e \in E, x \in e, u \in e \\ \frac{1}{q}, & \text{others} \end{cases}, \tag{5}$$

where p is the return parameter which controls the likelihood of immediately revisiting the previous vertex u in the walk and q is the in-out parameter which controls the walker to explore remote or close vertices. Thus the unnormalized transition probability of the second-order random walk model can be given as

$$\pi_2(x|v,u) = \alpha(x|v,u)\pi_1(x|v). \tag{6}$$

The above strategy works based on the relationship between parameters p, q, and 1. When $q < \min(p, 1)$, the walker is more inclined to visit vertices which are far away from vertex u. Such behavior is a reflection of *DFS* which encourages the random walker to explore further. In contrast, when $p < \min(q, 1)$ or $1 < \min(p, q)$, the walker tends to visit the previous vertex u or the vertices close to vertex u. Such walks obtain a local view with respect to the start vertex and approximate *BFS* behavior.

4.3 Degree Biased Random Walk

Based on the second-order random walk strategy introduced above, we can guide the walker to DFS or BFS, but some problems exist. Previous studies have shown that DFS and BFS search strategies will introduce particular negative bias. For example, it is observed that BFS introduces a bias toward vertices with big degrees [4, 22, 25]. On one hand, it is valuable for us to correct the negative bias for random walks [21]. On the other hand, it is also helpful for us to introduce some positive bias to capture some specific characteristics of networks or balance the update opportunities for vertices [12, 46]. However, based on the second-order model, we cannot control the walker to select a vertex of bigger degree or smaller degree, which has a great influence on random walks in hyper-networks.

Based on the above considerations, a degree biased random walk model is proposed in our work. There are two sampling strategies: *Big-Degree Biased Search (BDBS)*, where the walker tends to choose the neighbors with bigger degree as the next vertex, and *Small-Degree Biased Search (SDBS)*, where the walker tends to choose the neighbors with smaller degree as the next vertex. A simple illustration of BDBS and SDBS is shown in Figure 2.

To guide the random walks toward BDBS or SDBS, we introduce a bias coefficient $\beta(x)$ based on the degree of x with a parameter r. The key is to use r to control the coefficient rate between two vertices. To achieve the above goal, we use the following formula for BDBS:

$$\beta(x) = d(x) + r \quad (r > 0). \tag{7}$$

The bias coefficient rate of vertices x_1 and x_2 is calculated as follows:

$$\frac{\beta(x_1)}{\beta(x_2)} = \frac{d(x_1) + r}{d(x_2) + r} \quad (r > 0). \tag{8}$$

Obviously, $\beta(x)$ grows linearly with $d(x)$, and as r increases, bias coefficient rate $\beta(x_1)/\beta(x_2)$ is smoother than $d(x_1)/d(x_2)$, which means the degree of BDBS tendency declines.

The strategy of SDBS ($r < 0$) can be defined similarly to BDBS. Combining the cases together, the final formula is produced as follows:

$$\beta(x) = \begin{cases} d(x) + r, & r > 0 \\ \frac{1}{d(x) - r}, & r < 0 \\ 1, & r = 0 \end{cases}. \tag{9}$$

Combining with the previous random walk model, the final transition probability $p_2'(x|v,u)$ is defined as follows:

$$p_2'(x|v,u) = \frac{\alpha(x|v,u) \cdot \beta(x) \cdot \sum_{e \in E} w(e) \frac{h(v,e)}{d(v)} \frac{h(x,e)}{\delta(e)}}{Z}, \tag{10}$$

where $\alpha(\cdot)$ is the second-order search bias and Z is a normalizing factor. Based on the transition probability $p_2'(x|v,u)$, we can sample a sequence of vertices in a hyper-network.

4.4 The Hyper2vec Algorithm

The complete algorithm flow of Hyper2vec is shown in Algorithm 1. The algorithm consists of three main components: firstly, a preprocessing procedure; secondly, a random walk generator; and thirdly, an update procedure.

The preprocessing procedure handles the hyper-network and generates a probability matrix P based on the first-order random walk model in Section 4.1. The probability matrix is then guided by parameters p and q in the second-order random walk model in Section 4.2 and parameter r in the degree biased random walk model in Section 4.3.

The random walk generator generates a set of random walks. The modified transition probability is precomputed in the preprocessing procedure so that each step of random walk can be done efficiently in $O(1)$ time by using alias sampling [20] (a Python implementation is available online[1]).

The update procedure produces the final embedding result via the Skip-gram model. Skip-gram [24] is a language model which maximizes the co-occurrence probability among the words close to each other in sentences, which are replaced by random walks in our algorithm. Letting $f : V \rightarrow \mathbb{R}^d$ be a center mapping function from vertices to embeddings, $f' : V \rightarrow \mathbb{R}^d$ be a context mapping function, and $C(v)$ be the contexts of v, the optimization problem of Skip-gram for our model is formulated as follows:

$$\max_f \sum_{v \in V} \left(\sum_{c_i \in C(v)} \left(f(v) \cdot f'(c_i) - \log \sum_{u \in V} e^{f(v) \cdot f'(u)} \right) \right). \tag{11}$$

Complexity analysis: The time complexity of calculating the probability matrix P is $O(an)$, where n is the number of vertices and a is the average number of neighbors of each vertex which is usually small for real-world networks. For the biased second-order random walk procedure, it is useful to store the interconnections between the neighbors of every vertex, which incurs a time

[1] https://lips.cs.princeton.edu/the-alias-method-efficient-sampling-with-many-discrete-outcomes/.

ALGORITHM 1: The Hyper2vec Algorithm

```
Input: hypergraph $G = (V, E, w)$, embedding size $d$, window size $k$, number of walks $t$, walk
       length $l$, return parameter $p$, in-out parameter $q$, bias parameter $r$;
Output: matrix of vertex representations $\Phi \in \mathbb{R}^{|V| \times d}$;
Initialize $walks$ to $empty$;
Calculate transition probability matrix $P \in \mathbb{R}^{|V| \times |V|}$ according to Equation (4);
for $u \in V$ do
    for $v \in N_G(u)$ do
        $\Pi'_{\cdot uv} = \beta(v) \cdot P_{uv}$;
        for $x \in N_G(v)$ do
            $\Pi'_{uvx} = \alpha(x|v, u) \cdot \beta(x) \cdot P_{vx}$;
for $i$ from 1 to $t$ do
    for $v \in V$ do
        $walk$ = RandomWalk($G, \Pi', v, l$);
        Append $walk$ to $walks$;
$\Phi$ = Skip-gram($k, d, walks$);

Function $RandomWalk(G, \Pi', s, l)$
    Initialize $walk$ to $[s]$;
    Initialize $u$ to $null$;
    for $i$ from 2 to $l$ do
        $v$ = the last element of $walk$;
        $x$ = AliasSample($N_G(v), \Pi'_{uv}$);
        Append $x$ to $walk$;
        $u = v$;
    return $walk$;
```

complexity of $O(a^2n)$. The random walk procedure costs $O(tln)$ time, where t is the number of walks per vertex and l is the walk length. The Skip-gram model can be calculated efficiently by using negative sampling.

4.5 NHNE: Nonuniform Hyper-Network Embedding with Dual Mechanism

Based on the above model, we can capture various features of vertices in a hyper-network. However, for the hyperedges, we only consider the information that one hyperedge connects some vertices, which is low-order information. Some previous studies [2, 43] have demonstrated that more detailed edge features (e.g., interaction activities) can be utilized to enhance the performance of network embedding models.

For hyper-networks considered in this work, edge features are even more important as the hyperedges usually have some specific meanings and can be considered as a similar identity to the vertices. For instance, the hyperedges of a co-author network represent the articles. In Hyper2vec, we use biased second-order random walks to preserve the high-order proximity and structural properties of vertices. In order to capture more information of the original hyper-networks, it is of importance to preserve the high-order proximity and structural properties of hyperedges.

To this end, as shown in Figure 3, we can reverse the role of vertices and hyperedges to build a dual hyper-network. Since the dual of a hyper-network is also a hyper-network, we can use the same strategy above to generate the dual random walks. The embedding of the dual hyper-network

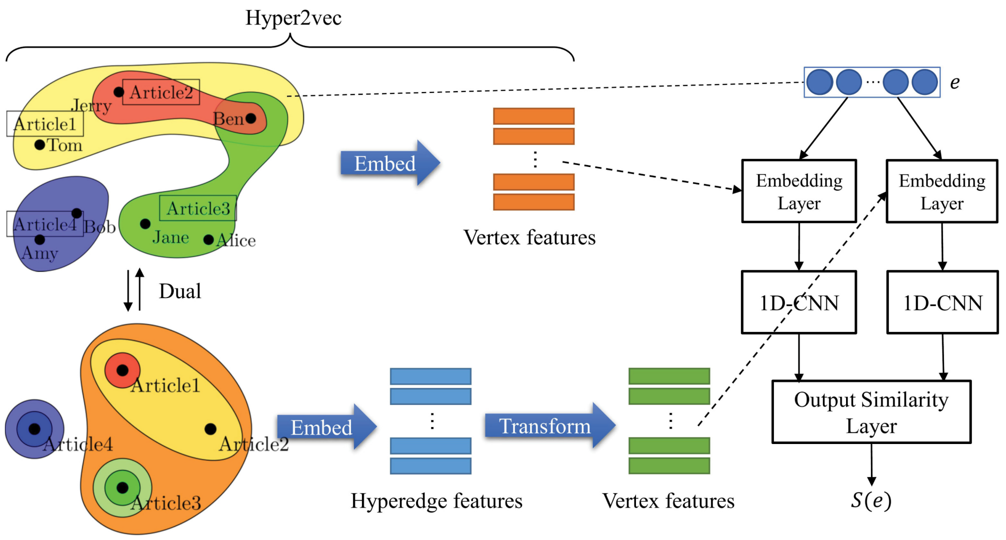


Fig. 3. The structure of NHNE. The two leftmost figures are the original hyper-network and the dual hyper-network. In this case, the original hyper-network is a co-author network (upper left) and the dual is an article network (lower left). Ben is the author of Article1–3, so there is a three-degree hyperedge incident with Article1–3 (orange). Jane and Alice are both the authors of Article3 only, so there are two one-degree hyperedges incident with Article3 (green). We use the embeddings learned by Hyper2vec from the original hyper-network and dual hyper-network as the initial weights of the two embedding layers, respectively. We output the final embeddings of vertices from the embedding layers and obtain a tuplewise similarity function to evaluate the tuplewise relationships of vertices.

corresponds to the hyperedge embedding of the original network. In order to get the embedding of vertex v, we can calculate the average embedding of the hyperedges incident with v. We use $f^*(v)$ to represent the embedding of v learned from dual hyper-network, which is calculated as follows:

$$f^*(v) = \frac{1}{\sum_{e \in E_G(v)} w(e)} \sum_{e \in E_G(v)} w(e) \cdot f(e), \tag{12}$$

where $f(e)$ is the embedding of hyperedge e learned from the dual hyper-network and $E_G(v)$ is the set of hyperedges incident with v. The procedure of the dual model can be done simultaneously with the original model.

The embeddings generated from the original hyper-network and the dual are complementary, which we will prove in our later experiments. To combine the original embedding and the dual embedding and capture tuplewise similarity, we introduce the proposed Nonuniform Hyper-Network Embedding with Dual Mechanism (NHNE) model. The framework of NHNE is shown in Figure 3.

In order to use the hyperedges as the inputs to the neural network, we first need to encode the hyperedges. A hyperedge is a set of vertices, so a straightforward way is to use the one-hot encoding of the vertices in the set as the encoding of the hyperedge. We use $X_e \in \mathbb{R}^{n \times z}$ to represent the encoding of hyperedge e, where n is the number of vertices and z is the max edge degree. The column vectors are in the order of the vertices and supplement zero vector when $\delta(e) < z$. Combining with the weights of the embedding layer (denoted as $W^{(emb)}$), we have $W^{(emb)} X_e \in \mathbb{R}^{d \times z}$ being the initial matrix representation of hyperedge e. We use the embeddings learned by Hyper2vec (both original and dual) as the initial weights of the embedding layer, which is usually a good starting point for training.

Table 1. Statistics of the Datasets

Datasets	|V|	|E|	$ave(\delta(e))$	$max(\delta(e))$
DBLP	7,995	18,364	3.02	29
IMDb	4,423	4,334	2.89	3

As discussed above, hyper-networks are usually nonuniform, which means that the degree of hyperedges is not fixed. A hyperedge can be treated as a vertex sequence, but the order of the vertices has no meaning, thus it is suitable to model with *1D-CNNs*. The 1D-CNN denoted as $c(\cdot)$ consists of 1D convolution layer and max-pooling layer, which takes $W^{(emb)}X_e \in \mathbb{R}^{d\times z}$ as input and outputs the latent vector representation of e with dimension d.

With the dual hyper-network mechanism, we first use two 1D-CNNs to capture the hidden information of the original and dual hyper-networks, respectively, and then concatenate them as the input of the output similarity layer. Finally, the tuplewise similarity function $S(\cdot)$ is defined as follows:

$$S(e) = \sigma(W^{(out)}[c_o(W_o^{(emb)}X_e); c_d(W_d^{(emb)}X_e)] + b), \tag{13}$$

where $\sigma(\cdot)$ is the sigmoid function; $W_o^{(emb)}$ and $W_d^{(emb)}$ denote the weights of the original and dual embedding layer, respectively; $c_o(\cdot)$ and $c_d(\cdot)$ denote the original and dual 1D-CNN, respectively; $W^{(out)}$ denotes the weights of the output similarity layers; and b is the bias. The loss function of NHNE is defined as follows:

$$\mathcal{L} = \frac{1}{|E|}\sum_{e\in E}\left(-\log S(e) + \sum_{e_n\in E_n}\log S(e_n)\right), \tag{14}$$

where E_n is the negative samples of e. For each $e_n \in E_n$, $e_n \notin E$ and $|e_n| = |e|$. RMSProp [16] is used to train model parameters. Finally, we output and concatenate the embeddings of vertices from the embedding layers and obtain a tuplewise similarity function to evaluate the tuplewise relationships of vertices. Comparing with the embeddings learned by Hyper2vec, the embeddings updated by NHNE contain more features about the tuplewise relationships among vertices.

5 EXPERIMENTS

In this section, we evaluate the performance of our model. We apply our model to two real-world hyper-networks on link prediction and vertex classification tasks.

5.1 Datasets

We use DBLP and IMDb datasets in our experiments. The statistics of the datasets are listed in Table 1, where $ave(\delta(e))$ is the average edge degree, and $max(\delta(e))$ is the maximal edge degree. DBLP[2] is a computer science bibliography. The DBLP hyper-network is produced from the DBLP XML data taken on April 6, 2019. The dataset contains information about a collection of papers and their authors, so we can abstract each author as a vertex and the co-author relationship of each article as a hyperedge in the past 10 years. We focus on several conferences in three different research fields: NIPS and ICML from machine learning; CVPR and ICCV from computer vision; and ACL, EMNLP, and NAACL from natural language processing. Authors whose degrees are less than three are filtered out, and finally, the hyper-network contains 7,995 vertices (authors) and 18,364 hyperedges (publications).

[2] http://dblp.uni-trier.de/xml/.

The IMDb[3] dataset includes the information that actors co-starred in movies. In our experiments, we use a subset of the IMDb dataset. Each actor is represented as a vertex and the top three actors in each movie form a hyperedge. We filter out the actors with less than three degrees in the same way as DBLP, and finally, the hyper-network contains 4,423 vertices (actors) and 4,334 hyperedges (movies).

5.2 Experiment Setup

Our experiments evaluate the performance of Hyper2vec and NHNE on two learning tasks: link prediction for hyperedges and classification for vertices. For each task, we compare the performance of Hyper2vec and NHNE against the following network embedding algorithms.

- DeepWalk [27]: This approach learns d-dimensional embeddings by simulating uniform random walks of fixed length from all the vertices in normal networks.
- Node2vec [14]: This approach learns d-dimensional embeddings by simulating the second-order random walks in normal networks, which is an extension of DeepWalk.
- LINE [33]: This approach learns d-dimensional embeddings by optimizing the first-order proximity and the second-order proximity objective functions separately and then concatenate the embeddings trained by the two methods for each vertex.
- AROPE [47]: This approach supports shifts across arbitrary orders with a low marginal cost based on singular value decomposition (SVD) framework.
- Metapath2vec [11]: This approach learns d-dimensional embeddings for heterogeneous information networks by meta-path-based random walks, which is applied to hyper-networks with the star expansion.
- BiNE [13]: This approach learns d-dimensional embeddings for bipartite networks by stimulating biased random walks, which also can be applied to hyper-networks with the star expansion.
- HHE [49]: This approach learns d-dimensional representations for vertices in hyper-networks by solving eigenvector problems related to hyper-network Laplacian matrices.
- HGE [45]: This approach learns d-dimensional representations for vertices in hyper-networks by incorporating multi-way relations into an optimization problem related to geometric mean and arithmetic mean.

Some algorithms above are conventional pairwise network embedding methods. To deal with the tasks in hyper-networks by traditional algorithms, in our experiments, we use clique expansion [32] to transform a hyper-network into a normal network. BiNE is designed for the bipartite network embedding, which can be applied to hyper-networks with the star expansion [1]. HHE and HGE are the recently hyper-network embedding models, which can be applied to uniform hyper-networks without a conversion to normal networks. In order to apply these two models on nonuniform hyper-networks, we pad empty vertices to the hyperedge whose degree is smaller than the max degree.

Parameter Setting: We have a fair parameter tuning for all methods. For random walk–based models like DeepWalk, Node2vec, and Hyper2vec, we set the window size as 5, walk length as 20, and number of walks as 10. For the second-order and biased random walk model, we do a grid search over $p, q \in \{0.25, 0.5, 1, 2, 4\}$ and $r \in \{0, \pm 1, \pm 2, \pm 4, \pm 8, \pm 16\}$. For LINE, we set the number of negative samples as five. For AROPE, we range the order from 1 to 20 and choose the best one. For BiNE and HGE, we follow the default setting and turn some important parameters like learning rate and epochs. For NHNE, we set the number of units of the hidden layers as 32 and the size

[3]https://www.imdb.com/.

of the convolution kernel as 3. We uniformly set the embedding size as 32 for all methods. The experiments were repeated five times.

5.3 Link Prediction

In link prediction, we are given a network with part of the edges removed, and the task is to predict these removed edges based on the resultant network. For hyper-networks, the above problem is raised to predict the missing hyperedges.

There are some binary operations to measure the pairwise relationship of two vertices based on their embeddings. Given two vertices u and v with embeddings $\mathbf{x} \in \mathbb{R}^d$ and $\mathbf{y} \in \mathbb{R}^d$, the L1 distance (L1D), the L2 distance (L2D), and the cosine similarity (COS) metrics are respectively defined as follows:

$$L1D(\mathbf{x},\mathbf{y}) = \sum_{i=1}^{d} |x_i - y_i|, \tag{15}$$

$$L2D(\mathbf{x},\mathbf{y}) = \sqrt{\sum_{i=1}^{d} (x_i - y_i)^2}, \tag{16}$$

$$COS(\mathbf{x},\mathbf{y}) = \frac{\sum_{i=1}^{d} x_i y_i}{\sqrt{\sum_{i=1}^{d} x_i^2} \cdot \sqrt{\sum_{i=1}^{d} y_i^2}}. \tag{17}$$

To measure the tuplewise relationship, we calculate the mean among all pairwise relationship metric values in a candidate hyperedge. And tuplewise relationships with lower L1/L2 distance or higher cosine similarity are more likely to be true. Here we should mention that different models are suitable for different metrics, so we use the maximum of the three metrics to evaluate and compare the performance for each model.

For the DBLP dataset, we randomly remove 50 percent of existing hyperedges as the positive testing samples and use the remaining network to train the model and get the embedding of each vertex. To generate the same number of negative samples, each time, we sample several vertices from the vertex set uniformly to form a tuple. And then we judge whether this tuple is a hyperedge of the original hyper-network. If so, we ignore it and generate a new tuple until the tuple is not in the hyperedge set. The degree distribution of negative samples is the same as the degree distribution of removed hyperedges. For the IMDb dataset, we only remove 30 percent of existing hyperedges due to the sparsity of the network, otherwise, the structure of the network may not be well preserved and network connectivity will also be significantly affected. The Area Under Curve (AUC) scores are shown in Table 2.

By comparing Hyper2vec and NHNE with other models, our observations are as follows:

- Hyper2vec performs better than all baselines, which shows the generality of the biased second-order random walk model.
- Combining with the dual mechanism and tuplewise similarity function, NHNE has an impressive performance boost. This indicates that the embeddings generated by the original model and the dual model are complementary to each other and shows the superiority of the tuplewise similarity function.
- Although BiNE, HHE, and HGE are also designed to be hyper-networks, BiNE is just suitable to predict the links in the bipartite networks, and HHE and HGE have difficulty handling the nonuniform hyper-networks, which illustrates the irreplaceability of our model.

In short, with a biased second-order random walk model, Hyper2vec performs better than most other models. And benefiting from the combination of dual hyper-network, NHNE has a further significant improvement on link prediction tasks.

Table 2. AUC Scores of Link Prediction on DBLP and IMDb

Embedding Algorithm	**DBLP**				**IMDb**			
	L1D	*L2D*	*COS*	***MAX***	*L1D*	*L2D*	*COS*	***MAX***
DeepWalk	0.9499	0.9512	0.9471	0.9512	0.8656	0.8699	0.5774	0.8699
Node2vec	0.9510	0.9529	0.9489	0.9529	0.8696	0.8732	0.5761	0.8732
LINE	0.9358	0.9303	0.9186	0.9358	0.6719	0.6524	0.4725	0.6719
AROPE	0.4624	0.4656	0.8866	0.8866	0.2899	0.2919	0.7154	0.7154
Metapath2vec	0.9505	0.9515	0.9456	0.9515	0.8014	0.8045	0.6212	0.8045
BiNE	0.4498	0.4454	0.3939	0.4498	0.3527	0.3422	0.2108	0.3527
HHE	0.5882	0.5883	0.5978	0.5978	0.6930	0.6817	0.6655	0.6930
HGE	0.8279	0.8297	0.8160	0.8297	0.8218	0.8244	0.8175	0.8244
Hyper2vec	0.9534 ± 0.0010	0.9548 ± 0.0006	0.9511 ± 0.0011	0.9548**	0.8742 ± 0.0026	0.8775 ± 0.0022	0.6177 ± 0.0072	0.8775*
NHNE	**0.9631**** ± 0.0005				**0.8904**** ± 0.0007			

Significantly outperforms Node2vec (Hyper2vec) or Hyper2vec (NHNE) at the **, 0.01 and *, 0.05 level by paired t-test.

Table 3. Results of Vertex Classification on DBLP and IMDb

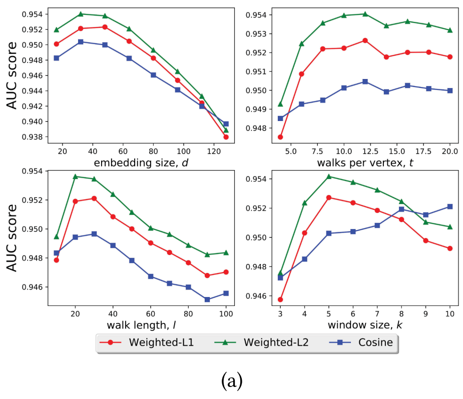
Embedding Algorithm	**DBLP**		**IMDb**	
	Micro-F1	***Macro-F1***	***Micro-F1***	***Macro-F1***
DeepWalk	0.8746	0.8703	0.4423	0.1525
Node2vec	0.8766	0.8721	0.4441	0.1542
LINE	0.8143	0.8080	0.4403	0.1244
AROPE	0.8143	0.8061	0.4439	0.1125
Metapath2vec	0.8735	0.8695	0.4406	0.1321
BiNE	0.4161	0.2123	0.4309	0.0929
HHE	0.4176	0.1982	0.4341	0.0859
HGE	0.5247	0.4471	0.4354	0.1240
Hyper2vec	0.8866** ± 0.0013	0.8825** ± 0.0013	0.4482* ± 0.0026	0.1583* ± 0.0054
NHNE	**0.8939**** ± 0.0012	**0.8893**** ± 0.0012	**0.4535**** ± 0.0029	**0.1750**** ± 0.0049

Significantly outperforms Node2vec (Hyper2vec) or Hyper2vec (NHNE) at the **, 0.01 and *, 0.05 level by paired t-test.

5.4 Vertex Classification

In the task of vertex classification, every vertex of the hyper-network has one or more labels. When the number of labels of a vertex is more than one, we specifically call it multi-label classification.

The DBLP dataset is collected from three different domains of computer science, so we can simply divide the publications into three categories. We count the number of publications in three categories for each author, and then pick the highest frequency category as the label. For the IMDb dataset, there are multiple genres in each movie. We firstly remove some genres with very small frequencies and then count the number of movies in these genres for each actor. Due to the large number of genres, there is no clear division for actors, so we choose the top three genres of each actor as the actor's labels according to the number of genres associated with each actor. Logistic regression is used as the outer classifier and Micro-F1 and Macro-F1 are used to evaluate the performance of the models. The results of fivefold cross-validation are shown in Table 3.



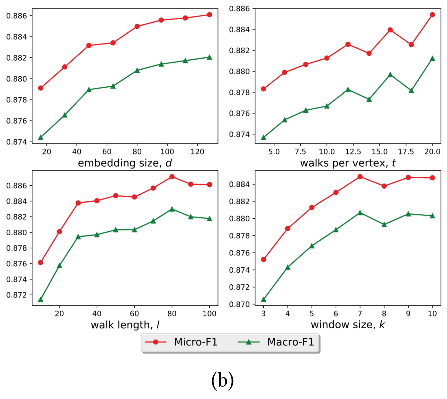


Fig. 4. Parameter sensitivity of first-order random walk on link prediction (a) and vertex classification (b).

By comparing Hyper2vec and NHNE with other models, our observations, which are basically consistent with the results of the link prediction, are as follows:

—Hyper2vec performs better than all baselines on classification tasks, which shows the flexibility of the biased second-order random walk model.
—By combining the features of dual hyper-networks and updating the embeddings with neural network model, NHNE has an impressive performance boost. This again indicates that the embeddings generated by the original model and the dual model are complementary.
—Similar to link prediction, BiNE, HHE, and HGE perform badly on classification tasks, which shows that these models cannot learn the representations of nonuniform hyper-networks well.

All in all, based on the biased second-order random walk model, Hyper2vec achieves the goal of flexibility. With the dual mechanism and the framework of 1D-CNN, NHNE achieves the goal of merging the features of the dual hyper-network and has significant performance improvements for different tasks on different datasets.

5.5 Parameter Sensitivity

In this section, we examine the parameter sensitivity of Hyper2vec for link prediction and vertex classification tasks on the DBLP dataset.

We first analyze the parameter sensitivity of the first-order random walk model for link prediction (Figure 4(a)). For each parameter, we fix other parameters as default. From the figure, we find that the effect of parameters on L1 distance and L2 distance is consistent, and L2 distance is usually better than L1 distance. For the parameter sensitivity of the first-order random walk model on classification tasks drawn in Figure 4(b), the general trend is that larger parameters will achieve better results. This is because compared with our default setting, larger parameters will get more training time. In fact, the walk length, the number of walks per vertex, and the number of epochs of the Skip-gram model ensure the adequacy of model training, and the window size determines the maximum distance that the current vertex has associated with the vertex, which is affected by the datasets and the metrics. In short, for the first random walk model, the most important thing is to ensure that the model is adequately trained and choose an appropriate window size.

Secondly, we analyze the parameters of the second-order random walk model by drawing the heat maps (Figure 5 and Figure 6). The abscissa is the logarithm of p, and the ordinate is the logarithm of q (use 2 as the base of the logarithm) with the optimal first-order random walk parameter

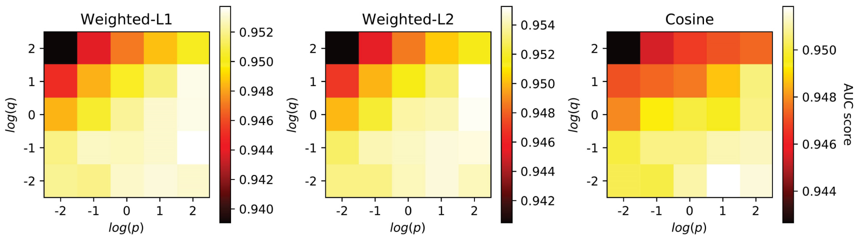


Fig. 5. Parameter sensitivity of second-order random walk on link prediction.

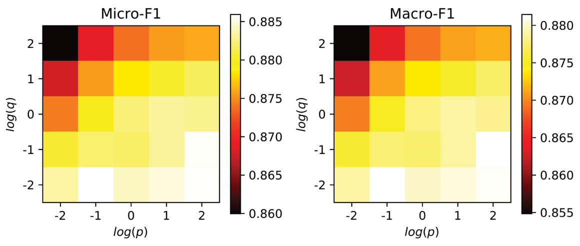


Fig. 6. Parameter sensitivity of second-order random walk on vertex classification.

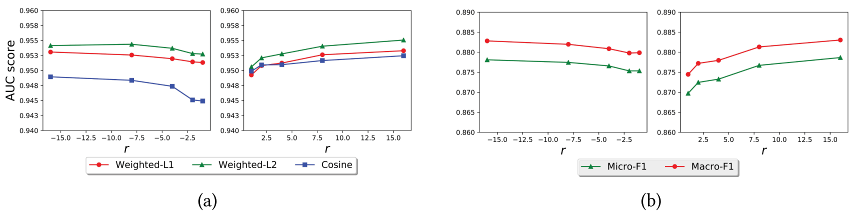


Fig. 7. The influence of degree bias parameter r on link prediction (a) and vertex classification (b).

setting. From the figure, we find that it is better to set $p > 1$ and $q < 1$, which shows that random walks which have a DFS tendency perform better on all tasks and metrics.

Finally, we evaluate the influence of bias parameter r (Figure 7). We find that the DBLP hyper-network doesn't need too much degree tendency, and this is different with the IMDb hyper-network where we set $r = 4$. In conclusion, the second-order random walk and the degree biased random walk model make the random walk more flexible. On the one hand, they can correct the parameters of the previous random walk model; on the other hand, they can bring possible performance improvement.

All in all, to deal with the parameter setting, we are firstly free to set these parameters based on the prior knowledge of the underlying task and domain at no additional cost. We may also tackle the problem as a semi-supervised problem to learn the best set of parameters with very few labeled data. In addition, we should mention that although the parameter settings are optimal, Hyper2vec cannot achieve the performance of NHNE. This implies that the original hyper-network does not capture all the information, and further verifies that the features captured from the dual hyper-network are supplement to the original hyper-network.

5.6 Effect of the Weight Initialization

In this section, we use different models to learn the initial embeddings as the initial weights of the embedding layer. We design several variations of NHNE and conduct experiments on link

Table 4. AUC Scores of Link Prediction on DBLP for Variations

Algorithm	***Pairwise***	***Tuplewise***
DeepWalk′	0.9512	0.9543
Node2vec′	0.9529	0.9558
LINE′	0.9358	0.9402
AROPE′	0.8866	0.8889
Metapath2vec′	0.9515	0.9546
BiNE′	0.4498	0.9105
HHE′	0.5978	0.8492
HGE′	0.8297	0.8632
Random′	-	0.9178
Hyper2vec′	0.9548	0.9572
Dual2vec′	0.9532	0.9533
NHNE	-	**0.9631**

prediction with the DBLP dataset:

— Hyper2vec′: This model uses only one 1D-CNN layer to update the embeddings learned by the Hyper2vec algorithm on the original hyper-network and train the tuplewise similarity function. This is to say, compared with NHNE, Hyper2vec′ lacks the dual mechanism.
— Dual2vec′: This model uses only one 1D-CNN layer to update the embeddings learned by the Hyper2vec algorithm on the dual hyper-network and train the tuplewise similarity function.
— Random′: This model uses only one 1D-CNN layer and has no weights initialization.
— (Baseline)′: These models use the embeddings learned by the baselines to initialize the weights of the embedding layer (similar to Hyper2vec′ and without the dual mechanism).

The AUC scores are shown in Table 4 (Tuplewise column). We also list the AUC scores of the pairwise metrics from Table 2 as a comparison in the Pairwise column. From the table, we have the following conclusions:

— Comparing the AUC scores in the Tuplewise column with the Pairwise column, we can determine that the tuplewise similarity function is necessary for hyper-network on link prediction tasks.
— Comparing Hyper2vec′ with other baselines, we find that a better tuplewise score is usually accompanied by a better pairwise score, which implies the importance of the initialization of the embedding layers.
— Compared with Hyper2vec′ and Dual2vec′, NHNE has a significant performance improvement, which implies that the features learned by the original hyper-network and dual hyper-network are complementary.

To explore the impact of embedding layer updates, we freeze the two embedding layers of NHNE as a comparison, which gets an AUC score of 0.9605. Comparing with NHNE, we find that the embedding update has a certain effect, for it incorporates some features of the tuplewise relationships to final embeddings.

6 DISCUSSION AND CONCLUSION

In this article, we first propose a flexible model named Hyper2vec to learn the embeddings of hyper-networks. Our model introduces a biased second-order random walk strategy to hyper-networks in the framework of Skip-gram. Secondly, due to the generality of Hyper2vec, it is helpful and feasible for us to combine the hyperedge features in dual hyper-networks to build a 1D-CNN–based model called NHNE, which has a significant improvement and beats all the state-of-the-art models in different tasks. For future work, we plan to explore deeper relationships among parameters, prediction tasks, and the structure of hyper-networks. Future extensions of our model may include a smarter parameter adjustment mechanism. In addition, we can also generate biased random walks in a local view, i.e., dynamically adjust the bias to capture the local properties. In addition, it's also possible for us to try other bias metrics, e.g., local clustering coefficient and incorporate more information of hyper-networks, e.g., the attributes of vertices and hyperedges.

REFERENCES

[1] Sameer Agarwal, Kristin Branson, and Serge Belongie. 2006. Higher order learning with graphs. In *ICML*. ACM, 17–24. DOI: https://doi.org/10.1145/1143844.1143847

[2] Lars Backstrom and Jure Leskovec. 2011. Supervised random walks: Predicting and recommending links in social networks. In *WSDM*. ACM, 635–644. DOI: https://doi.org/10.1145/1935826.1935914

[3] Inci M. Baytas, Cao Xiao, Fei Wang, Anil K. Jain, and Jiayu Zhou. 2018. Heterogeneous hyper-network embedding. In *ICDM*. IEEE, 875–880.

[4] Luca Becchetti, Carlos Castillo, Debora Donato, Adriano Fazzone, and I. Rome. 2006. A comparison of sampling techniques for web graph characterization. In *LinkKDD*.

[5] Mikhail Belkin and Partha Niyogi. 2002. Laplacian eigenmaps and spectral techniques for embedding and clustering. In *NIPS*. 585–591.

[6] Claude Berge. 1984. *Hypergraphs: Combinatorics of Finite Sets*. Vol. 45. Elsevier.

[7] Claude Berge and Edward Minieka. 1973. Graphs and hypergraphs. North-Holland, Amsterdam.

[8] Shaosheng Cao, Wei Lu, and Qiongkai Xu. 2016. Deep neural networks for learning graph representations. In *AAAI*. 1145–1152.

[9] Shiyu Chang, Wei Han, Jiliang Tang, Guo-Jun Qi, Charu C. Aggarwal, and Thomas S. Huang. 2015. Heterogeneous network embedding via deep architectures. In *KDD*. ACM, 119–128.

[10] Peng Cui, Xiao Wang, Jian Pei, and Wenwu Zhu. 2018. A survey on network embedding. *IEEE Transactions on Knowledge and Data Engineering* (2018).

[11] Yuxiao Dong, Nitesh V. Chawla, and Ananthram Swami. 2017. metapath2vec: Scalable representation learning for heterogeneous networks. In *KDD*. ACM, 135–144.

[12] Rui Feng, Yang Yang, Wenjie Hu, Fei Wu, and Yueting Zhang. 2018. Representation learning for scale-free networks. In *AAAI*.

[13] Ming Gao, Leihui Chen, Xiangnan He, and Aoying Zhou. 2018. BiNE: Bipartite network embedding. In *SIGIR*. 715–724.

[14] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *KDD*. ACM, 855–864. DOI: https://doi.org/10.1145/2939672.2939754

[15] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural collaborative filtering. In *WWW*. 173–182.

[16] Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. 2012. Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. Retrieved on January 20, 2019 from http://www.cs.toronto.edu/~hinton/coursera/lecture6/lec6.pdf.

[17] Peter D. Hoff, Adrian E. Raftery, and Mark S. Handcock. 2002. Latent space approaches to social network analysis. *Journal of the American Statistical Association* 97, 460 (2002), 1090–1098. DOI: https://doi.org/10.1198/016214502388618906

[18] Jie Huang, Chuan Chen, Fanghua Ye, Jiajing Wu, Zibin Zheng, and Guohui Ling. 2019. Hyper2vec: Biased random walk for hyper-network embedding. In *DASFAA*. Springer, 273–277.

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

[20] Richard A. Kronmal and Arthur V. Peterson, Jr. 1979. On the alias method for generating random variables from a discrete distribution. *The American Statistician* 33, 4 (1979), 214–218.

[21] Maciej Kurant, Athina Markopoulou, and Patrick Thiran. 2010. On the bias of BFS (breadth first search). In *ITC*. IEEE, 1–8. DOI: https://doi.org/10.1109/ITC.2010.5608727

[22] Sang Hoon Lee, Pan-Jun Kim, and Hawoong Jeong. 2006. Statistical properties of sampled networks. *Physical Review E* 73, 1 (2006), 016102. DOI: https://doi.org/10.1103/physreve.73.016102

[23] David Liben-Nowell and Jon Kleinberg. 2007. The link-prediction problem for social networks. *Journal of the Association for Information Science and Technology* 58, 7 (2007), 1019–1031.

[24] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

[25] Marc Najork and Janet L. Wiener. 2001. Breadth-first crawling yields high-quality pages. In *WWW*. ACM, 114–118. DOI: https://doi.org/10.1145/371920.371965

[26] Andrew Y. Ng, Michael I. Jordan, and Yair Weiss. 2002. On spectral clustering: Analysis and an algorithm. In *NIPS*. 849–856.

[27] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *KDD*. ACM, 701–710. DOI: https://doi.org/10.1145/2623330.2623732

[28] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2009. BPR: Bayesian personalized ranking from implicit feedback. In *UAI*. AUAI Press, 452–461.

[29] Sam T. Roweis and Lawrence K. Saul. 2000. Nonlinear dimensionality reduction by locally linear embedding. *Science* 290, 5500 (2000), 2323–2326.

[30] Michael Schlichtkrull, Thomas N. Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. 2018. Modeling relational data with graph convolutional networks. In *ESWC*. Springer, 593–607.

[31] Chuan Shi, Binbin Hu, Wayne Xin Zhao, and S. Yu Philip. 2018. Heterogeneous information network embedding for recommendation. *IEEE Transactions on Knowledge and Data Engineering* 31, 2 (2018), 357–370.

[32] Liang Sun, Shuiwang Ji, and Jieping Ye. 2008. Hypergraph spectral learning for multi-label classification. In *KDD*. ACM, 668–676. DOI: https://doi.org/10.1145/1401890.1401971

[33] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. Line: Large-scale information network embedding. In *WWW*. 1067–1077. DOI: https://doi.org/10.1145/2736277.2741093

[34] Lei Tang and Huan Liu. 2011. Leveraging social media networks for classification. *Data Mining and Knowledge Discovery* 23, 3 (2011), 447–478.

[35] Joshua B. Tenenbaum, Vin De Silva, and John C. Langford. 2000. A global geometric framework for nonlinear dimensionality reduction. *Science* 290, 5500 (2000), 2319–2323.

[36] Ke Tu, Peng Cui, Xiao Wang, Fei Wang, and Wenwu Zhu. 2018. Structural deep embedding for hyper-networks. In *AAAI*.

[37] Daixin Wang, Peng Cui, and Wenwu Zhu. 2016. Structural deep network embedding. In *KDD*. ACM, 1225–1234. DOI: https://doi.org/10.1145/2939672.2939753

[38] Hongwei Wang, Fuzheng Zhang, Min Hou, Xing Xie, Minyi Guo, and Qi Liu. 2018. Shine: Signed heterogeneous information network embedding for sentiment link prediction. In *WSDM*. ACM, 592–600.

[39] Suhang Wang, Jiliang Tang, Charu Aggarwal, Yi Chang, and Huan Liu. 2017. Signed network embedding in social media. In *SDM*. SIAM, 327–335.

[40] Carolyn Watters and Michael A. Shepherd. 1990. A transient hypergraph-based model for data access. *ACM Transactions on Information Systems (TOIS)* 8, 2 (1990), 77–102.

[41] George H. Weiss. 1983. Random walks and their applications: Widely used as mathematical models, random walks play an important role in several areas of physics, chemistry, and biology. *American Scientist* 71, 1 (1983), 65–71.

[42] Fei Wu, Ya-Hong Han, and Yue-Ting Zhuang. 2010. Multiple hypergraph clustering of web images by mining-word2image correlations. *Journal of Computer Science and Technology* 25, 4 (2010), 750–760. DOI: https://doi.org/10.1007/s11390-010-9362-9

[43] Biao Xiang, Ziqi Liu, Jun Zhou, and Xiaolong Li. 2018. Feature propagation on graph: A new perspective to graph representation learning. *arXiv preprint arXiv:1804.06111*.

[44] Linchuan Xu, Xiaokai Wei, Jiannong Cao, and Philip S. Yu. 2017. Embedding of embedding (eoe): Joint embedding for coupled heterogeneous networks. In *WSDM*. ACM, 741–749.

[45] Chia-An Yu, Ching-Lun Tai, Tak-Shing Chan, and Yi-Hsuan Yang. 2018. Modeling multi-way relations with hypergraph embedding. In *CIKM*. ACM, 1707–1710. DOI: https://doi.org/10.1145/3269206.3269274

[46] Ziqian Zeng, Xin Liu, and Yangqiu Song. 2018. Biased random walk based social regularization for word embeddings. In *IJCAI*. 4560–4566.

[47] Ziwei Zhang, Peng Cui, Xiao Wang, Jian Pei, Xuanrong Yao, and Wenwu Zhu. 2018. Arbitrary-order proximity preserved network embedding. In *KDD*. ACM, 2778–2786. DOI: https://doi.org/10.1145/3219819.3219969

[48] Denny Zhou, Jiayuan Huang, and Bernhard Schölkopf. 2007. Learning with hypergraphs: Clustering, classification, and embedding. In *NIPS*. 1601–1608.

[49] Yu Zhu, Ziyu Guan, Shulong Tan, Haifeng Liu, Deng Cai, and Xiaofei He. 2016. Heterogeneous hypergraph embedding for document recommendation. *Neurocomputing* 216 (2016), 150–162. DOI: https://doi.org/10.1016/j.neucom.2016.07.030

Received July 2019; revised March 2020; accepted March 2020