Chapter 6
Emerging Topics of Heterogeneous Graph Representation

Abstract Heterogeneous graph (HG) embedding, aiming to project HG into a low-dimensional space, has attracted considerable research attention. We have introduced some kinds of HG embedding methods, and there are also some other essential topics on HG embeddings. In this chapter, we will introduce three novel HG embedding methods. Specifically, to learn semantic-preserving and robust node representations, we study the problem of adversarial learning on HG. Also, we work with large-scale heterogeneous interaction graphs and focus on the problem of importance sampling on HG embedding. Moreover, we explore the intrinsic spaces of HG and propose a hyperbolic space based HG embedding method.

6.1 Introduction

Recently, some efforts begin to combine graph embedding methods with varied machine learning technologies to obtain better graph representations. These machine learning technologies include generative adversarial networks (GANs) [12, 26], importance sampling [5, 17], as well as hyperbolic representation [20, 22]. Specifically, GANs hinge on the idea of adversarial learning, in which discriminators and generators compete with each other to learn better underlying data distributions. Besides, importance sampling aims to reduce the computational and memory costs in machine learning. Moreover, hyperbolic spaces are non-Euclidean spaces, which are more suitable to model data with a hierarchical structure.

Some homogeneous graph embedding methods combined with these technologies have shown their effectiveness in modeling graphs. For example, GAN-based homogeneous graph embedding methods [6, 29] leverage generative and discriminative components to learn graph representation. The generative component aims to learn the underlying connectivity distribution in the graph, while the discriminative component aims to predict the probability of edge existence between nodes. Besides, to make large-scale graph representation learning possible, a natural idea is to sample a small set of important nodes from the whole graph. The importance

sampling strategies have achieved significant improvements for homogeneous graph representation learning [5, 14, 17, 39]. Moreover, some researchers begin to embed graphs into low-dimensional hyperbolic spaces. They find that embedding a graph in hyperbolic spaces would have a low distortion when the graph has a hierarchical structure [11, 22], and the hierarchical relation between nodes can be reflected by analyzing hyperbolic embedding [22].

In this chapter, we introduce three emerging HG embedding methods. First, **H**G **e**mbedding with **GAN**-based adversarial learning (named **HeGAN** [16]) is designed to learn semantic-preserving and robust node representations, which leverages the relation-aware discriminator and generator to fit the heterogeneous setting. Second, to reduce the cost of large-scale HG embedding, **Hete**rogeneous importance **Samp**ling (named **HeteSamp** [19]) is proposed, which introduces both type-dependent and type-fusion samplers with self-normalized and adaptive estimators to ensure the model efficiency. Third, assuming the underlying spaces of HG are hyperbolic spaces, **H**yperbolic **H**eterogeneous **N**etwork **E**mbedding (named **HHNE**) [33] is proposed to preserve the structure and semantic information of HG in hyperbolic spaces.

6.2 Adversarial Learning

6.2.1 Overview

GANs [12, 26] have been developed for learning robust latent representations in various applications [31, 36]. In recent researches, GANs only study homogeneous graphs [6, 24, 30, 37], such as the bibliographic data in Fig. 6.1a , so they do not consider the heterogeneity of nodes and edges, resulting in unsatisfactory performance on HG. In existing methods, positive (real) & negative (fake) nodes can only be differentiated by the network structure, which is shown in Fig. 6.1b. Therefore, it is urgent to design new formal discriminator and generator to distinguish and simulate the real and fake semantic-rich nodes involving various relationships. Generators are essentially picking an existing node from the original network according to the learned distribution, without the ability to generalize to "unseen" nodes. Not surprisingly, they do not generate the most representative fake nodes, as such nodes may not even appear in the network. Thus, it is important to design a generator that can efficiently produce latent fake samples.

We exploit the heterogeneity of HG in adversarial setting in order to learn semantic-preserving and robust node representations. Different from previous efforts [2, 6, 24, 30, 37], HeGAN raises two major novelties to address the challenges of adversarial learning on HG. It is not only relation-aware to capture rich semantics, but also equipped with a generalized generator that is effective and efficient. It proposes a new form of discriminator and generator, as illustrated in Fig. 6.1c. To a given relation, the discriminator can tell whether a node pair is real or fake, whereas the generator can produce fake node pairs that mimic real pairs. To further

improve the effectiveness and efficiency of sample generation, we propose a generalized generator, which is able to directly sample latent nodes from a continuous distribution.

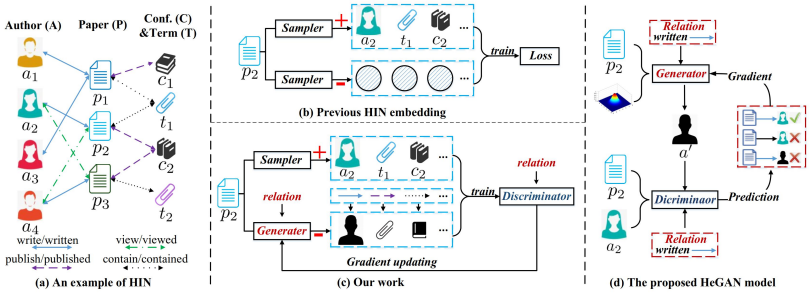


Fig. 6.1 Overview of HeGAN. (a) A toy example of HG for bibliographic data. (b) An example of previous HG embedding. (c) Relation-aware discriminator and generator in our work. (d) The framework of HeGAN for adversarial learning on HGs.

6.2.2 The HeGAN Model

6.2.2.1 Model Framework

HeGAN is based on GAN so that it consists of a discriminator and a generator. It is a novel framework for HG embedding with GAN-based adversarial learning. The discriminator and generator are relation-aware in HeGAN. The discriminator is able to tell apart the specified relationship between nodes, and the generator will try to generate nodes which has certain relation with other nodes. The node pair is considered true in the following cases: (i) It is a positive pair based on network topology; (ii) The pair is formed under the correct relation. HeGAN provides a generalized generator, which is able to directly sample latent nodes from a continuous distribution, such that (i) no softmax computation is necessary; and (ii) fake samples are not restricted to the existing nodes.

Existing studies typically model the distribution of nodes using some forms of softmax over all nodes in the original network. In terms of effectiveness, their fake samples are limited to the nodes in the network, whereas the most representative fake samples may fall "in between" the existing nodes in the embedding space. For the bibliographic data in Fig. 6.1a, given a paper p_2, it can only choose fake samples from $\mathcal{V}$, such as a_1 and a_3. However, both may not be adequately similar to real samples such as a_2. Towards a better sample generation, HeGAN introduces a generalized generator that can produce latent nodes not appearing in the embedding

space. The generator can sample fake nodes directly without using a softmax. The framework of HeGAN is in Fig. 6.1d.

6.2.2.2 Relation-aware Discriminator

In an HG, the relation-aware discriminator $D(e_v|u,r;\theta^D)$ evaluates the relation r between the pair of nodes u and v. Specifically, $u \in \mathcal{V}$ is a given node and $r \in \mathcal{R}$ is a given relation from a HG $\mathcal{G}$, e_v is the embedding of a sample node v (which can be real or fake), and θ^D denotes the model parameters of D. In essence, D outputs a probability that the sample v is connected to u under the relation r. We quantify this probability as:

$$D(e_v|u,r;\theta^D) = \frac{1}{1+\exp\left(-e_u^{D\top} M_r^D e_v\right)}, \tag{6.1}$$

where $e_v \in \mathbb{R}^{d\times 1}$ is the embedding of the sample v, $e_u^D \in \mathbb{R}^{d\times 1}$ is the learnable embedding of node u, and $M_r^D \in \mathbb{R}^{d\times d}$ is a learnable relation matrix for relation r. $\theta^D = \{e_u^D : u \in \mathcal{V}, M_r^D : r \in \mathcal{R}\}$ forms the model parameters of D, *i.e.,* the union of all node embeddings and relation matrices learnt by D. Naturally, the probability should be high when v is a positive sample related to u through r , or low when it is a negative sample. In general, a sample v forms a triple $\langle u,v,r\rangle$ together with the given u and r , and each triple belongs to one of the three cases below with regard to its polarity. Each case also contributes to one part of the discriminator loss, inspired by the idea of conditional GAN [26].

Case 1: Connected under a given relation. That is, nodes u and v are indeed connected through the right relation r on the HG $\mathcal{G}$, such as $\langle a_2, p_2, \text{write}\rangle$ shown in Fig. 6.1a. Such a triple is considered positive and can be modeled by the below loss.

$$\mathcal{L}_1^D = \mathbb{E}_{\langle u,v,r\rangle\sim P_\mathcal{G}} - \log D(e_v^D|u,r). \tag{6.2}$$

Here we draw the positive triple from the observed $\mathcal{G}$, denoted as $\langle u,v,r\rangle \sim P_\mathcal{G}$.

Case 2: Connected under an incorrect relation. That is, u and v are connected on the HG under a wrong relation $r' \neq r$, such as $\langle a_2, p_2, \text{view}\rangle$. The discriminator is expected to mark them as negative, as their connectivity does not match the desired semantics carried by the given relation r. We define this part of loss as follows:

$$\mathcal{L}_2^D = \mathbb{E}_{\langle u,v\rangle\sim P_\mathcal{G}, r'\sim P_{\mathcal{R}'}} - \log\left(1 - D(e_v^D|u,r')\right). \tag{6.3}$$

Here, we still draw the pair of nodes $\langle u,v\rangle$ from $\mathcal{G}$, but the negative relation r' is drawn from $\mathcal{R}' = \mathcal{R}\setminus\{r\}$ uniformly.

Case 3: Fake node from a relation-aware generator. That is, given a node $u \in \mathcal{V}$, it can form a fake pair with the node v supplied by the generator $G(u,r;\theta^G)$, such as $\langle a', p_2, \text{write}\rangle$ in Fig. 6.1d. The generator is also relation-aware: It attempts to generate a fake node's embedding that mimics the real nodes connected to u under

the correct relation r. Again, the discriminator aims to identify this triple as negative, which can be formulated as follows.

$$\mathcal{L}_3^D = \mathbb{E}_{\langle u,r\rangle\sim P_{\mathcal{G}},e'_v\sim G(u,r;\theta^G)} - \log\left(1 - D(e'_v|u,r)\right). \tag{6.4}$$

Note that the fake sample v's embedding e'_v is drawn from the generator G's learnt distribution. On the other hand, the discriminator D simply treats e'_v as non-learnable input, and only optimizes its own parameters θ^D.

The loss function of the discriminator is comprehensive by the above three parts:

$$\mathcal{L}^D = \mathcal{L}_1^D + \mathcal{L}_2^D + \mathcal{L}_3^D + \lambda^D \|\theta^D\|_2^2, \tag{6.5}$$

where $\lambda_D > 0$ controls the regularization term to avoid overfitting. The parameters θ^D of the discriminator can be optimized by minimizing $\mathcal{L}^D$.

6.2.2.3 Relation-aware Generalized Generator

Given a node $u \in \mathcal{V}$ and a relation $r \in \mathcal{R}$, the generator $G(u,r;\theta^G)$ aims to generate a fake node v likely to connect to u in the context of relation r. In other words, v should be as close as possible to a real node on $\mathcal{G}$, in which has $\langle u,v,r\rangle \sim P_{\mathcal{G}}$. On the other hand, the generator is generalized, which means the fake node v can be latent and not found in $\mathcal{V}$.

To meet the two requirements, the generator should also employ relation-specific matrices (for relation-awareness), and generate samples from an underlying continuous distribution (for generalization). One of the suitable distributions is Gaussian distribution:

$$\mathcal{N}(e_u^{G\top} M_r^G, \sigma^2 I), \tag{6.6}$$

where $e_u^G \in \mathbb{R}^{d\times 1}$ and $M_r^G \in \mathbb{R}^{d\times d}$ denote the node embedding of $u \in \mathcal{V}$ and the relation matrix of $r \in \mathcal{R}$ for the generator. In other words, it is a Gaussian distribution with mean $e_u^{G\top} M_r^G$ and covariance $\sigma^2 I \in \mathbb{R}^{d\times d}$ for some choices of σ. Intuitively, the mean represents a fake node is likely to be connected to u by relation r, and the covariance represents potential deviations. As neural networks have shown strong ability in modeling complex structure [15], we integrate the multi-layer perceptron (MLP) into the generator for enhancing the expression of the fake samples. Hence, our generator is formulated as follows,

$$G(u,r;\theta^G) = f(W_L \cdots f(W_1 \boldsymbol{e} + b_1) + b_L), \tag{6.7}$$

where we draw e from the distribution $\mathcal{N}(e_u^{G\top} M_r^G, \sigma^2 I)$. Here W_* and b_* respectively denote the weight matrix and the bias vector for each layer, and f is an activation function. The parameter set of the generator is thus $\theta^G = \{e_u^G : u \in \mathcal{V}, M_r^G : r \in \mathcal{R}, W_*, b_*\}$, *i.e.,* the union of all node embeddings and relation matrices, as well as the parameters of MLP.

As motivated earlier, the generator aims to fool the discriminator by generating close-to-real fake samples, such that the discriminator gives high score to them.

$$\mathcal{L}^G = \mathbb{E}_{\langle u,r\rangle \sim P_{\mathcal{G}}, e'_v \sim G(u,r;\theta^G)} - \log D(e'_v|u,r) + \lambda^G \|\theta^G\|_2^2, \tag{6.8}$$

where $\lambda^G > 0$ controls the regularization term. The parameters θ^G of the generator can be optimized by minimizing $\mathcal{L}^G$.

Algorithm 6.1 Model training for HeGAN.

Input: HG $\mathcal{G}$, number of generator and discriminator trainings per epoch n_G, n_D, number of samples n_s

Initialize θ_G and θ_D for G and D, respectively
while not converge **do**
  for $n = 0; n < n_D$ **do** ▷ **Discriminator training**
    Sample a batch of triples, *i.e.,* $\langle u, v, r\rangle \sim P_{\mathcal{G}}$
    Generate n_s fake nodes $e'_v \sim G(u, r; \theta^G)$ for each $\langle u, r\rangle$
    Sample n_s relations $r' \sim P_{\mathcal{R}'}$ for each $\langle u, v\rangle$
    Update θ^D according to Eq. 6.5
  end for
  for $n = 0; n < n_G$ **do** ▷ **Generator training**
    Sample a batch of triples, *i.e.,* $\langle u, v, r\rangle \sim P_{\mathcal{G}}$
    Generate n_s fake nodes $e'_v \sim G(u, r; \theta^G)$ for each $\langle u, r\rangle$
    Update θ^G according to Eq. 6.8
  end for
end while
return θ^G and θ^D

6.2.2.4 Model Training

We adopt the iterative optimization strategy to train HeGAN. The iterative optimization methods follow the steps below: At each iteration, θ^G is fixed first, and then the generator generates fake samples to optimize θ^D and thus improve the discriminator. Next, θ^D is fixed, followed by optimizing θ^G in order to produce increasingly better fake samples as evaluated by the discriminator. The above steps repeat until the model converges. The model training for HeGAN is outlined in Algorithm 6.1.

6.2.3 Experiments

6.2.3.1 Experimental Settings

We conduct extensive experiments on three benchmark datasets [10, 15], namely DBLP (with four types of nodes: *Author*, *paper*, *conference* and *term*), Yelp (*user*, *business*, *service*, *star* and *reservation*), AMiner (*author*, *paper*, *conference* and

reference). We organize them into HGs, as summarized in Table 6.1. We perform node classification and link prediction on three datasets.

Table 6.1 Description of datasets.

Datasets	#Nodes	#Edges	#Node types	#Labels
DBLP	37,791	170,794	4	4
Yelp	3,913	38,680	5	3
Aminer	312,776	599,951	4	6

For the baselines, we consider three categories of network embedding methods: Traditional (Deepwalk [25], LINE [28]), GAN-based (GraphGAN [30], ANE [6]) and HG (HERec-HNE [27], HIN2vec [10], Metapath2vec [8]) embedding algorithms.

6.2.3.2 Node Classification

We use 80% of the labeled nodes to train a logistic regression classifier, and test the classifier on the remaining 20% of the nodes. We report *Accuracy* on the test set in Table 6.2.

Table 6.2 Performance comparison on node classification. (bold: best; underline: runner-up).

Methods	DBLP			Yelp			AMiner		
	Micro-F1	Macro-F1	Accuracy	Micro-F1	Macro-F1	Accuracy	Micro-F1	Macro-F1	Accuracy
Deepwalk	0.9201	0.9242	0.9298	0.8262	0.7551	0.8145	0.9519	0.9460	0.9529
LINE-1st	0.9239	0.9213	0.9285	0.8229	0.7440	0.8126	0.9776	0.9713	0.9788
LINE-2nd	0.9144	0.9172	0.9236	0.7591	0.5518	0.7571	0.9469	0.9341	0.9471
GraphGAN	0.9198	0.9210	0.9286	0.8098	0.7268	0.7820	-	-	-
ANE	0.9143	0.9153	0.9189	0.8232	0.7623	0.7932	0.9256	0.9203	0.9221
HERec-HNE	0.9214	0.9228	0.9299	0.7962	0.7713	0.7912	0.9801	0.9726	0.9784
HIN2vec	0.9141	0.9115	0.9224	0.8352	0.7610	0.8200	0.9799	0.9775	0.9801
Metapath2vec	0.9288	0.9296	0.9360	0.7953	0.7884	0.7839	0.9853	0.9860	0.9857
HeGAN	**0.9381****	**0.9375****	**0.9421****	**0.8524****	**0.8031****	**0.8432****	**0.9864***	**0.9873***	**0.9883***

In node classification, HeGAN consistently outperforms the best baseline with statistical significance. It is also worth noting that our performance margins over the best baseline become smaller compared to the node clustering task, since in classification all methods are helped by the supervision, narrowing their gaps.

6.2.3.3 Link Prediction

In this task, we predict *user-business* links on Yelp, and *author-paper* links on DBLP and AMiner. We randomly hide 20% of such links from the original network as the ground truth positives, and randomly sample disconnected node pairs of the given form as negative instances. The ground truth serves as our test set. We adopt two

Table 6.3 Performance comparison on link prediction. (bold: best; underline: runner-up).

	Methods	DBLP			Yelp			AMiner		
		Acc	AUC	F1	Acc	AUC	F1	Acc	AUC	F1
Logistic Regression	Deepwalk	0.5441	0.5630	0.5208	0.7161	0.7825	0.7182	0.4856	0.5182	0.4618
	LINE-1st	0.6546	0.7121	0.6685	0.7226	0.7971	0.7099	0.5983	0.6413	0.6080
	LINE-2nd	0.6711	0.6500	0.6208	0.6335	0.6745	0.6499	0.5604	0.5114	0.4925
	GraphGAN	0.5241	0.5330	0.5108	0.7123	0.7625	0.7132	-	-	-
	ANE	0.5123	0.5430	0.5280	0.6983	0.7325	0.6838	0.5023	0.5280	0.4938
	HERec-HNE	0.7123	0.7823	0.6934	0.7087	0.7623	0.6923	0.7089	0.7776	0.7156
	HIN2vec	0.7180	0.7948	0.7006	0.7219	0.7959	0.7240	0.7142	0.7874	0.7264
	Metapath2vec	0.5969	0.5920	0.5698	0.7124	0.7798	0.7106	0.7069	0.7623	0.7156
	HeGAN	**0.7290****	**0.8034****	**0.7119****	**0.7240****	**0.8075****	**0.7325****	**0.7198****	**0.7957****	**0.7389****
Inner Product	Deepwalk	0.5474	0.7231	0.6874	0.5654	0.8164	0.6953	0.5309	0.6064	0.6799
	LINE-1st	0.6647	0.7753	0.7363	0.6769	0.7832	0.7199	0.6113	0.6899	0.7123
	LINE-2nd	0.4728	0.4797	0.6325	0.4193	0.7347	0.5909	0.5000	0.4785	0.6666
	GraphGAN	0.5532	0.6825	0.6214	0.5702	0.7725	0.6894	-	-	-
	ANE	0.5218	0.6543	0.6023	0.5432	0.7425	0.6324	0.5421	0.6123	0.6623
	HERec-HNE	0.5123	0.7473	0.6878	0.5323	0.6756	0.7066	0.6063	0.6912	0.6798
	HIN2vec	0.5775	0.8295	0.6714	0.6273	**0.8340**	0.4194	0.5348	0.6934	0.6824
	Metapath2vec	0.4775	0.6926	0.6287	0.5124	0.6324	0.6702	0.6243	0.7123	0.6953
	HeGAN	**0.7649****	**0.8712****	**0.7837****	**0.7391****	0.8298	**0.7705****	**0.6505****	**0.7431****	**0.7752****

ways to perform link prediction, namely, *logistic regression* and *inner product*. The evaluation metrics consist of *Accuarcy*, *AUC* and *F1*, as shown in Table 6.3.

We observe that, the performance of HeGAN over the best baseline is much larger with inner product than with logistic regression. It is hypothesized that HeGAN innately learns a much better structure and semantics preserving embedding space than the baseline methods, since the inner product only relies on the learnt representations without resorting to any external supervision.

The complete method and more experiments can be found in [16].

6.3 Importance Sampling

6.3.1 Overview

To make large-scale graph representation learning possible, researchers have proposed several sampling strategies on Graph Neural Networks (GNNs), including node-wise neighborhood sampling [14, 35] and layer-wise neighborhood sampling [5, 17, 39] to accelerate the training process of GNNs by reducing the number of edges in computation. The former is to sample neighbors for each node while the latter is to sample neighbors from the whole graph. Unfortunately, both the node- and layer-wise sampling only deal with homogeneous graphs, which are inadequate in many real-world scenarios such as E-commerce graphs, as they (1) deal with homogeneous graphs only; (2) take all the nodes of a graph as initial candidates; (3) still incur rapidly increasing cost with more layers. There are two challenges we have to face with as follows.

First, how to design an effective sampler that works with the heterogeneous neighborhoods? While a sampler can be easily defined on a per-node, per-type basis, it is not clear how we can sample neighborhoods in a batch, given that the candidates consist of different types of nodes. Two alternatives could work, namely, **type-dependent** and **type-fusion** sampling. In the former, a sampler is deployed for each neighbor type, and these type-based samplers sample from their respective sub-neighborhood on their own. In the latter, a single sampler is deployed, which treats the entire common neighborhood of the batch as its candidates. Intuitively, the type-fusion sampler would work better, as it takes the influences of total types of interactions into account and models the sampling distribution over all types jointly.

Second, how to design the corresponding effective estimators with the sampled heterogeneous neighborhoods? With the neighbors being sampled based on the global importance in a batch, traditional importance sampling could introduce unwanted variance because of the imbalance between the local importance to the given target node and the global importance to the given batch. To address this challenge, we propose **self-normalized estimators** and **adaptive estimators**, respectively. The former is to adjust the estimators by self-normalizing importance of sampled neighborhoods while the latter is to automatically learn the global importance (i.e., the importance distribution of candidate neighborhoods) by modeling both structural and attributed information.

Finally, we test the proposed sampling strategies on two public datasets, and the experimental results demonstrate that the proposed framework can achieve statistically significant improvements. Compared with the full model without any sampling, we achieve a memory cost reduction by up to 92.48% and a time cost reduction by up to 85.95%, while maintaining the same level of accuracy on two real-world datasets.

6.3.2 The HeteSamp Model

This section introduces the architecture of the proposed HeteSamp model, which consists of the the general **H**eterogeneous **I**nteraction **G**raph **E**mbedding (named **HIGE**) and the heterogeneous sampling framework. The heterogeneous sampling framework leverages type-dependent and type-fusion strategies to overcome the expensive time cost and high computational complexity. Notice that, we name the large-scale heterogeneous graph as heterogeneous interaction graph to emphasize the rapidly increasing scale. Moreover, as HeteSamp consists of multiple samplers and estimators, we name the specific strategies according to their adopted sampler and estimator.

6.3.2.1 The General HIGE

To tackle with HIGs, a general idea is to reconstruct node embeddings by propagating information from its heterogeneous neighbors, and backwardly propagate

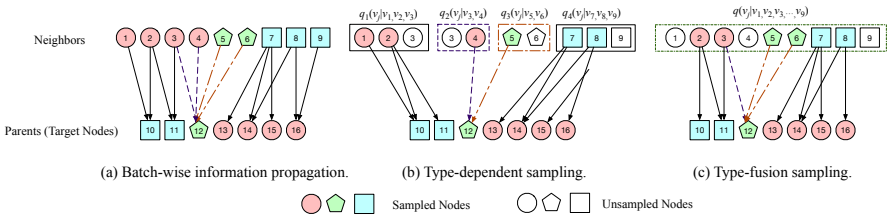


Fig. 6.2 The sampling processes of type-dependent and type-fusion sampling.

gradients [32, 38, 4], as shown in Fig. 6.2a. Specifically, given a node v_i with type $\phi(v_i)$ and its edges $\{e_{v_i,v_j,r}|v_j \in \mathcal{N}_{v_i,r}, r \in \mathcal{R}\}$, the aggregated information of node v_i from type-r neighborhoods is

$$\boldsymbol{g}'_{v_i,r} = \sum_{v_j \in \mathcal{N}_{v_i,r}} \frac{1}{|\mathcal{N}_{v_i,r}|} w(v_i, v_j, r) \boldsymbol{h}_{v_j}, \tag{6.9}$$

where $\boldsymbol{g}'_{v_i,r}$ is the aggregated information, $\mathcal{N}_{v_i,r}$ is the type-r neighborhoods of node v_i, $v_j \in \mathcal{N}_{v_i,r}$ is a specific type-r neighbor of node v_i, $w(v_i, v_j, r)$ denotes the weight of edge $e_{v_i,v_j,r}$, $\boldsymbol{h}_{v_j} \in \mathbb{R}^d$ denotes the embedding of node v_j, d is the dimension of $\boldsymbol{h}_{v_j}$. $w(v_i, v_j, r)$ between node v_i and v_j is calculated by

$$w(v_i, v_j, r) = \sigma(\boldsymbol{x}_{v_i,v_j,r} \boldsymbol{\lambda}_r + b_r), \tag{6.10}$$

where σ is an activation function, $\boldsymbol{x}_{v_i,v_j,r} \in \mathbb{R}^{d_r}$ is the edge features between node v_i and v_j, d_r is the length of type-r edge features, $\boldsymbol{\lambda}_r \in \mathbb{R}^{d_r \times 1}$ and $b_r \in \mathbb{R}$ are the weight and bias parameters shared by type-r edges.

We then take all semantics into account by aggregating information from different-typed neighborhoods and constructing the global embedding $\boldsymbol{h}'_{v_i} \in \mathbb{R}^d$ of node v_i as

$$\boldsymbol{h}'_{v_i} = \sigma\left(concat(\boldsymbol{g}'_{v_i,r_0}, \boldsymbol{g}'_{v_i,r_1}, \cdots, \boldsymbol{g}'_{v_i,r_{|\mathcal{R}|}}) \boldsymbol{W}_{\phi(v_i)} + b_{\phi(v_i)}\right), \tag{6.11}$$

where σ is an activation function, $concat(\cdot)$ is the concatenation option, $\boldsymbol{W}_{\phi(v_i)} \in \mathbb{R}^{|\mathcal{R}|d \times d}$ denotes the projection matrix of type-$\phi(v_i)$ nodes, $b_{\phi(v_i)}$ denotes the bias and $|\mathcal{R}|$ denotes the total number of edge types $\mathcal{R}$ in the heterogeneous graph. The computational complexity of the general model is linear with the scale of edges and nodes on heterogeneous graphs.

6.3.2.2 Batch-Wise Heterogeneous Sampling

To reduce computational overhead and memory cost of the training process, a naïve idea is to sample several neighbors rather than aggregating information from all neighborhoods. Previous works [14, 35] on homogeneous works usually adopt node-wise sampling to sample several neighbors per node for learning. Based on the global and local structural information, current works [5, 17, 39] propose layer-wise sam-

pling strategies which consider the whole neighborhood as the candidates. However, these strategies are to deal with homogeneous graphs but ignore the abundant semantics of heterogeneous nodes and edges. Moreover, these strategies can only deal with small graphs because the overhead for node-wise and layer-wise sampling during optimization could be quite expensive, or even unaffordable.

In this section, Eq. (6.9) can be re-written as

$$\boldsymbol{g}'_{v_i,r} = \mathbb{E}_q \left[\frac{p(v_j|v_i,r)}{q(v_j|\boldsymbol{B}_k)} w(v_i,v_j,r) \boldsymbol{h}_{v_j} \right], \tag{6.12}$$

where $\boldsymbol{B}_k$ denotes the target nodes in the k^{th} mini-batch, $q(v_j|\boldsymbol{B}_k)$ denotes the corresponding sampling probability in the k^{th} mini-batch, or in other words, the importance in this batch, and $p(v_j|v_i,r)$ equals to $\frac{1}{|\mathcal{N}_{v_i,r}|}$. Thereby, we attempt to make heterogeneous sampling for each batch. The batch-wise heterogeneous sampling probability is defined as

$$\hat{v_j} \sim q(\hat{v_j}|v_1,v_2,\cdots,v_{|\boldsymbol{B}_k|}) \quad v_j \in \{\mathcal{N}_{v_i,r}|r \in \mathcal{R}, v_i \in \boldsymbol{B}_k\}, \tag{6.13}$$

where $\hat{v_j}$ is the sampled neighbor. As shown in Fig. 6.2a, we sample instances from the union neighborhood of all the target nodes in a batch, where the union neighborhood denotes the union of the individual neighbors of each target node. Since sampling is now done for each batch instead of each target node, batch-wise sampling is often a good choice to reduce computational overhead and memory cost of the training process. Compared with layer-wise sampling methods [5, 17, 39] which require the total nodes as candidates and have to load the whole graph structure before training, our batch-wise heterogeneous sampling focuses on the neighborhood union in the current batch. Compared with node-wise sampling which sample neighborhoods for each target node, our batch-wise heterogeneous sampling contains the advantages of reducing the overhead of sampling.

6.3.2.3 Type-Dependent Sampling Strategy

By now, we have defined the general batch-wise heterogeneous sampling. Different from traditional importance sampling, the neighborhoods of each batch connect with target nodes based on different-typed interactions. A straightforward idea is to design a sampler for each type, respectively. For example, in Fig. 6.2b, there are four types of candidates, and we sample neighbors from each type of candidates respectively by using one sampler. By adopting type-dependent sampling, we consider $q(v_j|v_1,v_2,\cdots,v_{|\boldsymbol{B}_k|})$ as a set of $\{q_r(v_j|\cdot)|r \in \mathbb{R}\}$, and sample type-$r$ neighborhoods with the corresponding sampler $q_r(v_j|\cdot)$. The remaining problem for type-dependent sampling is how to design the exact form of this sampler so as to keep low variance for efficient training. Here we define the average information $\boldsymbol{\mu}_{q_r}$ of $\boldsymbol{g}'_{i,r}$ as

$$\boldsymbol{\mu}_{q_r} = \frac{1}{|\boldsymbol{B}_k|} \sum_{v_i \in \boldsymbol{B}_k} \boldsymbol{g}'_{v_i,r} = \frac{1}{|\boldsymbol{B}_k|} \sum_{v_j} q_r(v_j|\cdot) \sum_{v_i \in \boldsymbol{B}_k} \frac{p(v_j|v_i,r)w(v_i,v_j,r)\boldsymbol{h}_{v_j}}{q_r(v_j|\cdot)}, \quad (6.14)$$

where $|\boldsymbol{B}_k|$ is the number of type-r samples. Then, the variance Var_{q_r} of $\boldsymbol{\mu}_{q_r}$ is calculated by

$$Var_{q_r}(\boldsymbol{\mu}_{q_r}) = \frac{1}{|\boldsymbol{B}_k|} \mathbb{E}_{q_r} \frac{\left[\boldsymbol{\mu}_{q_r} q_r(v_j|\cdot) - \sum_{v_i \in \boldsymbol{B}_k} p(v_j|v_i,r)w(v_i,v_j,r)\boldsymbol{h}_{v_i}\right]^2}{q_r(v_j|\cdot)^2}. \quad (6.15)$$

To ensure minimizing the variance, a better sampler is shown as

$$q_r(v_j) = \frac{\sum_{v_i \in \boldsymbol{B}_k} p(v_j|v_i,r)^2}{\sum_{v_j} \sum_{v_i \in \boldsymbol{B}_k} p(v_j|v_i,r)^2}. \quad (6.16)$$

6.3.2.4 Type-Fusion Sampling Strategy

Essentially, the type-dependent strategy pays attention to the influence of same-type neighborhoods, without jointly considering the effect of heterogeneous types. Thus, they only reduce the individual variance of each type, lacking a global picture on the overall variance of the batch. To address this weakness, we propose a type-fusion sampling strategy which considers the entire neighborhoods as the candidates.

The aggregation in Eq. (6.11) are to gather information from different-typed neighborhoods, and can be rewritten as

$$\boldsymbol{g}'_{v_i} = \sum_{r \in \mathcal{R}} \sum_{v_j \in \mathcal{N}_{v_i,r}} p(v_j|v_i,r)w(v_i,v_j,r)\boldsymbol{h}_{v_j}\boldsymbol{W}_r, \quad (6.17)$$

where $\boldsymbol{W}_r \in \mathbb{R}^{d \times d}$ denotes the relation-wise projection matrix. Under the type-fusion strategy, the average information in k-th batch over all types is given by

$$\boldsymbol{\mu}_q = \frac{1}{|\boldsymbol{B}_k|} \sum_{v_i \in \boldsymbol{B}_k} \boldsymbol{g}'_{v_i} = \frac{1}{|\boldsymbol{B}_k|} \sum_{v_j} q(v_j|\cdot) \sum_{r \in \mathcal{R}} \sum_{v_i \in \boldsymbol{B}_k} \frac{p(v_j|v_i,r)w(v_i,v_j,r)\boldsymbol{h}_{v_j}W_r}{q(v_j|\cdot)}, \quad (6.18)$$

As shown in Fig. 6.2c, the different-typed neighborhoods are sampled according to the type-fusion sampling distribution. Similar to that in Section 6.3.2.3, the corresponding sampler is defined as follows.

$$q_s(v_j) = \frac{\sum_{r \in \mathcal{R}} \sum_{v_i} p(v_j|v_i,r)^2}{\sum_{r \in \mathcal{R}} \sum_{v_j} \sum_{v_i \in \boldsymbol{B}_k} p(v_j|v_i,r)^2}, \quad (6.19)$$

where $q_s(v_j)$ is the structure-based type-fusion sampler.

While the above sampling strategies only consider link-based weight as the importance of nodes but ignoring the attributed-based importance on edges, we further pay attention to the edge features within HIGs and propose the corresponding sampler as follows

$$q_a(v_j) = \frac{\sum_{r\in\mathcal{R}}\sum_{v_i} p(v_j|v_i,r)f(\boldsymbol{x}(v_i,v_j,r))}{\sum_{r\in\mathcal{R}}\sum_{v_j}\sum_{v_i\in\boldsymbol{B}_k} p(v_j|v_i,r)f(\boldsymbol{x}(v_i,v_j,r))}, \tag{6.20}$$

where $q_a(v_j)$ denotes the adaptive sampler, $f(\boldsymbol{x}(v_i,v_j,r))$ denotes the importance of edge features which is calculated by $\sigma(\boldsymbol{x}(v_i,v_j,r)\boldsymbol{W}_{x,r})$ and $\boldsymbol{W}_{x,r}\in\mathbb{R}^{d_r\times 1}$ is the type-r learnable parameter.

6.3.2.5 Heterogeneous Self-Normalized and Adaptive Estimators

In type-dependent or type-fusion strategies, the batch-wise estimator is

$$\hat{\boldsymbol{g}}_{v_i} = \frac{1}{n}\sum_{r\in\mathcal{R}}\sum_{v_j\in\hat{\mathcal{N}}_{\boldsymbol{B}_k,r}} \frac{p(\hat{v}_j|\cdot)}{q(\hat{v}_j|\cdot)} w(v_i,\hat{v}_j,r)\boldsymbol{h}_{\hat{v}_j}\boldsymbol{W}_r, \tag{6.21}$$

where $\hat{\boldsymbol{g}}_{v_i}$ is the approximated information, n is the number of samples, $\hat{\mathcal{N}}_{\boldsymbol{B}_k,r}$ is the sampled type-r neighborhoods in this batch, $p(\hat{v}_j|\cdot)$ is equal to $\frac{1}{|\mathcal{N}_{v_i,r}|}$ and $q(\hat{v}_j|\cdot)$ denotes the heterogeneous samplers, such as q_r, q_s and q_a. However, this could increase the variance resulted from the imbalance of $p(\hat{v}_j|\cdot)$ and $q(\hat{v}_j|\cdot)$.

To address such problem, for structure-based $q_s(v_j)$, a promising way is to balance the weights based on self-normalized importance, similar to that in [23]. The corresponding estimator can be computed as

$$\hat{\boldsymbol{g}}_{sn,v_i} = \sum_{r\in\mathcal{R}}\sum_{\hat{v}_j\in\hat{\mathcal{N}}_{v_i,r}} \frac{\pi(\hat{v}_j)}{\sum_{\hat{v}'_j\in\hat{\mathcal{N}}_{v_i,r}}\pi(\hat{v}'_j)} w_{v_i,v_j,r}\boldsymbol{h}_j\boldsymbol{W}_r, \tag{6.22}$$

where $\hat{\boldsymbol{g}}_{sn,v_i}$ denotes the self-normalized information, $\pi(v_j) = \frac{p(v_j|\cdot)}{q(v_j|\cdot)}$, $\hat{\mathcal{N}}_r$ is the sampled neighborhoods of type r. Besides, for the adaptive sampling, we add the variance to the loss function and explicitly minimize the variance during training to achieve a better $q_a(v_j)$.

6.3.2.6 Optimization Framework

The overall loss function consists of four parts

$$L_k = L_{task,k} + \alpha L_{ep,k} + \beta\Omega(\boldsymbol{\Theta}) + \xi Var_{q_a,k}(\hat{\mu}_q), \tag{6.23}$$

where L_k is the loss value in k-th batch, $L_{task,k}$ is the loss from supervised learning in the same batch, $L_{ep,k}$ is the embedding propagation loss with sampling in the k-th batch, $Var_{q_a,k}(\hat{\mu}_q)$ is the variance of sampled information, $\Omega(\mathbf{\Theta})$ is the regularization of all latent parameters, and α, β and ξ are three hyper-parameters. Notice that, for type-dependent sampling and structure-based type-fusion sampling, ξ is 0.

6.3.3 Experiments

6.3.3.1 Experimental Settings

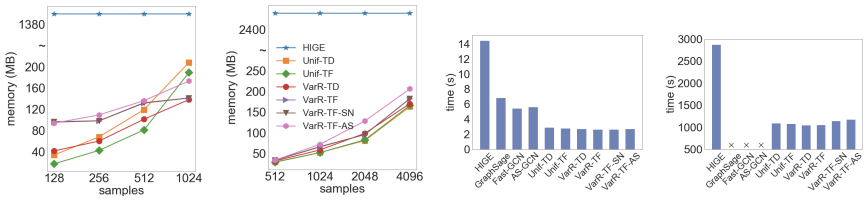
Datasets. We evaluate the empirical performance of our method on two real-world heterogeneous graphs, including the small Aminer graph consisting of 41,523 nodes and 199,429 edges, and the large Alibaba graph consisting of 4,527,222 nodes and 49,785,900 edges.

Baseline. We first compare our various importance sampling strategies with the general HIGE model and HAN [32] without any sampling. We also compare our model with state-of-the-art sampling algorithms on GCNs including Fast-GCN [17] and AS-GCN [5] to showcase our advantage of effectiveness and scalability, whereas the sampling-based models not only achieve superior accuracy, but also scale to large graphs. Finally, to study the performance of our proposed variance reduction, we also substitute our variance reduction sampler with a uniform sampler.

Parameter Settings. For all baselines and our methods, we set $\beta = 0.1$, $\gamma = 0.1$, $\psi = 0.1$, $\alpha = 0.4$. The maximum iteration is 100 for Aminer and 5 for Alibaba. We adopt Micro-F1 and Macro-F1 as the evaluation metrics for node classification on Aminer while adopting F1 and AUC as the evaluation metrics for link prediction on Alibaba. All the metrics are positively related to the performance of methods.

6.3.3.2 Empirical Validation

We perform empirical validation on the small Aminer graph for node classification and the much larger Alibaba interaction graph for link prediction. Notice that the non-HIGE-based baselines, GraphsAGE, Fast-GCN and AS-GCN are to deal with node classification, and these baselines can only work on Aminer; they cannot perform on Alibaba graph when the task is link prediction and this graph is too large. Furthermore, they are likely to suffer from insufficient memory on the large-scale Alibaba graph even if we implement the modified models for the task of link prediction.



(a) Aminer Memory (b) Alibaba Memory (c) Aminer Time (d) Alibaba Time

Fig. 6.3 Average memory cost and running time per iteration of training.

6.3.3.3 Effectiveness

We report the results in Table 6.4 and Table 6.5, respectively. We progressively sample more neighbors per batch and the sampling rate ranges from about 3% to 24% on smaller Aminer and 1.25% to 10% on the larger Alibaba graph. We make the following observations.

First, VarR-TF-AS generally achieves the best performance, whereas VarR-TF-SN comes as a close competitor. In particular, VarR-TF-AS performs as well as or even better than the original HIGE. This phenomenon is reasonable. On the one hand, HIGE aggregates information from all neighbors which may introduce noisy information while our sampling strategies are to sample valuable neighbors for training. On the other hand, we optimize the sampler by reducing the variance loss which enhances the similarity limitation between the sampled neighbors and its target nodes.

Second, compared to the corresponding uniform samplers, the variance reduction samplers guarantee more stable estimators and thus produce better results. In the VarR-* methods, type-fusion strategies outperform type-dependent methods, as the former consider all types jointly rather than independently, and reduce the variance of the whole batch rather than a single type.

Third, our proposed general HIGE can perform competitively with or even better than HAN. However, HAN has higher time complexity. For Alibaba dataset, the time cost of HAN (about 1 day) is quite larger than HIGE (about 5 hours).

Forth, all sampling strategies for HIGE perform significantly better than GCN-based models and GraphsAGE. On the one hand, sampling for HIGE takes the graph heterogeneity into consideration, whereas the two GCN-based models do not make use of such information. On the other hand, the sampling size or number of layers of Fast-GCN and AS-GCN may be not enough. However, even under current settings, Fast-GCN and AS-GCN are already several times slower than our method, as we shall see in the efficiency study.

Table 6.4 Micro/Macro-F1 scores for node classification on Aminer. Excluding HIGE and HAN, the best method is bolded and the second best is underlined.

	Micro-F1				Macro-F1			
Sampling size per batch	128 ~ 3%	256 ~ 6%	512 ~ 12%	1024 ~ 24%	128 ~ 3%	256 ~ 6%	512 ~ 12%	1024 ~ 24%
HIGE-Nil	0.1990				0.1961			
HIGE	0.9646				0.9593			
HAN	0.9512				0.9508			
GraphSAGE	0.2022	0.2039	0.2125	0.2119	0.1989	0.2023	0.2036	0.2073
Fast-GCN	0.2117	0.2244	0.2318	0.2361	0.1850	0.1898	0.2116	0.2119
AS-GCN	0.2361	0.2307	0.2390	0.2390	0.2005	0.2028	0.2004	0.2068
Unif-TD	0.3043	0.4123	0.4256	0.4221	0.2638	0.3907	0.3553	0.3962
Unif-TF	0.1780	0.2229	0.3785	0.6296	0.1266	0.1952	0.3081	0.5727
VarR-TD	0.8086	0.7990	0.8971	0.9123	0.7913	0.7894	0.8945	0.9133
VarR-TF	0.9461	0.9671	**0.9712**	0.9675	0.9424	0.9651	**0.9696**	0.9664
VarR-TF-SN	**0.9659**	0.9643	0.9612	0.9684	0.9637	0.9629	0.9631	0.9603
VarR-TF-AS	0.9650	**0.9676**	0.9705	**0.9687**	**0.9649**	**0.9667**	0.9602	**0.9671**

Table 6.5 F1 and AUC scores for purchase prediction on the Alibaba graph. Excluding HIGE and HAN, the best method is bolded and the second best is underlined.

	F1				ROC-AUC			
Sampling size per batch	512 ~ 1.25%	1024 ~ 2.5%	2048 ~ 5%	4096 ~ 10%	512 ~ 1.25%	1024 ~ 2.5%	2048 ~ 5%	4096 ~ 10%
HIGE-Nil	0.3994				0.5134			
HIGE	0.5663				0.7715			
HAN	0.5618				0.7704			
Unif-TD	0.4017	0.4226	0.4352	0.4371	0.5768	0.5826	0.5908	0.5924
Unif-TF	0.4008	0.4122	0.4125	0.4451	0.5731	0.5790	0.5862	0.5977
VarR-TD	0.4841	0.5003	0.5274	0.5682	0.6207	0.6504	0.6925	0.7475
VarR-TF	0.5769	0.5908	0.5709	0.5833	0.7648	**0.7671**	0.7653	0.7796
VarR-TF-SN	**0.5780**	0.5883	0.5802	0.5798	0.7669	0.7625	0.7660	0.7742
VarR-TF-AS	0.5729	**0.5913**	**0.5806**	**0.5844**	**0.7674**	0.7641	**0.7676**	**0.7799**

6.3.3.4 Efficiency

We first investigate the efficiency of our sampling strategies in memory cost. As shown in Fig. 6.3a and Fig. 6.3b, our models incur by up to 92.48% less memory. Further note that the differences in both the number of edges and memory cost are more prominent on the larger Alibaba graph, indicating excellent scalability of our sampling strategies. Second, in terms of the running time, as shown in Fig. 6.3c and Fig. 6.3d, VarR-TF-AS and VarR-TF-SN require less time than HIGE to attain close performance on the two datasets. Compared to HIGE, our models incur by up to 84.39% less time cost. Notice that when dealing with large-scale Alibaba dataset, the time cost is quite larger (about 1 day) than HIGE (about 5 hours).

The specific designs of HeteSamp can be found in [19].

6.4 Hyperbolic Representation

6.4.1 Overview

HG embedding has attracted considerable research attention recently. Most HG embedding methods choose Euclidean spaces to represent HG graphs, which is because Euclidean spaces are the natural generalization of our intuition-friendly, and visual three-dimensional space. However, a fundamental problem is that what are the appropriate or intrinsic underlying spaces of HGs? Therefore, we wonder whether Euclidean spaces are the intrinsic spaces of HGs?

Recently, hyperbolic spaces have gained momentum in the context of network science [20]. Hyperbolic spaces are spaces of constant negative curvature [3]. A superiority of hyperbolic spaces is that they expand faster that Euclidean spaces [22]. Therefore, it is easy to model complex data with low-dimensional embedding in hyperbolic spaces. Due to the characteristic of hyperbolic spaces, [20] assumes hyperbolic spaces underlie complex network and finds that data with power-law structure is suitable to be modeled in hyperbolic spaces. Also, some researchers begin to embed different data in hyperbolic spaces. For instance, [7] embeds text in hyperbolic spaces. [22] and [11] learn the hyperbolic embeddings of homogeneous networks. However, it is unknown whether HGs are suitable to be embedded in hyperbolic spaces.

In this section, we analyze the relation distribution in HGs and propose HHNE, which is able to preserve the structure and semantic information in hyperbolic spaces. HHNE leverages the meta-path guided random walk to generate heterogeneous neighborhoods to capture the structure and semantic relations in HGs. Then the proximity between nodes is measured by the distance in hyperbolic spaces. Also, HHNE is able to maximize the proximity between the neighborhood nodes while minimize the proximity between the negative sampled nodes. The optimization strategy of HHNE is derived to optimize hyperbolic embeddings iteratively.

6.4.2 The HHNE model

6.4.2.1 Model Framework

HHNE leverages the meta-path guided random walk to obtain neighbors for each node to capture the structure and semantic relations in HGs. Also, HHNE learns the embeddings by maximizing the proximity between the neighborhood nodes and minimizing the proximity between the negative sampled nodes. Moreover, we derive the optimization strategy of HHNE to upgrade the hyperbolic embeddings.

6.4.2.2 Hyperbolic HG Embedding

To design HG embedding methods in hyperbolic spaces, we makes use of the Poincaré ball model to describe hyperbolic spaces. Let $\mathbb{D}^d = \{x \in \mathbb{R}^d : \|x\| < 1\}$ be the *open* d-dimensional unit ball. The Poincaré ball model is defined by the manifold $\mathbb{D}^d$ equipped with the following Riemannian metric tensor $g_x^{\mathbb{D}}$:

$$g_x^{\mathbb{D}} = \lambda_x^2 g^{\mathbb{E}} \quad \text{where } \lambda_x := \frac{2}{1-\|x\|^2}, \tag{6.24}$$

where $x \in \mathbb{D}^d$, $g^{\mathbb{E}} = I$ denotes the Euclidean metric tensor.

HHNE aims to learn the representation of nodes to preserve the structure and semantic correlations in hyperbolic spaces. Given an HG $G = (V, E, T, \phi, \psi)$ with $|T_V| > 1$, HHNE is interested in learning the embeddings $\Theta = \{\theta_i\}_{i=1}^{|V|}, \theta_i \in \mathbb{D}^d$. HHNE preserves the structure by facilitating the proximity between a node and its neighborhoods. HHNE uses meta-path guided random walks [8] to obtain heterogeneous neighborhoods of a node. In meta-path guided random walks, the node sequences are restrained by the node types which are defined by meta-paths. Specifically, let t_{v_i} and t_{e_i} as the types of node v_i and edge e_i, respectively, given a meta-path $\mathcal{P} = t_{v_1} \xrightarrow{t_{e_1}} \dots t_{v_i} \xrightarrow{t_{e_i}} \dots \xrightarrow{t_{e_{n-1}}} t_{v_n}$, the transition probability at step i is defined as follows:

$$p(v^{i+1}|v_{t_{v_i}}^i, \mathcal{P}) = \begin{cases} \frac{1}{|N_{t_{v_{i+1}}}(v_{t_{v_i}}^i)|} & (v^{i+1}, v_{t_{v_i}}^i) \in E, \phi(v^{i+1}) = t_{v_{i+1}} \\ 0 & \text{otherwise,} \end{cases} \tag{6.25}$$

where $v_{t_{v_i}}^i$ is node $v \in V$ with type t_{v_i}, and $N_{t_{v_{i+1}}}(v_{t_{v_i}}^i)$ denotes the $t_{v_{i+1}}$ type of neighborhood of node $v_{t_{v_i}}^i$. The meta-path guided random walk strategy ensures that the semantic relationships between different types of nodes can be properly incorporated into HHNE.

In order to preserve the proximity between nodes and its neighborhoods in hyperbolic spaces, HHNE uses distances in Poincaré ball model to measure their proximity. Given nodes embeddings $\theta_i, \theta_j \in \mathbb{D}^d$, the distance in Poincaré ball is given by:

$$d_{\mathbb{D}}(\theta_i, \theta_j) = \cosh^{-1}\left(1 + 2\frac{\|\theta_i - \theta_j\|^2}{(1-\|\theta_i\|^2)(1-\|\theta_j\|^2)}\right). \tag{6.26}$$

It is worth noting that as the Poincaré ball model is defined in metric spaces, the distance in Poincaré ball meets the triangle inequality and can well preserve the transitivity in HG. Then, HHNE uses a probability to measure the node c_t is a neighborhood of node v as follows:

$$p(v|c_t; \Theta) = \sigma[-d_{\mathbb{D}}(\theta_v, \theta_{c_t})],$$

where $\sigma(x) = \frac{1}{1+\exp(-x)}$. Then the object of HHNE is to maximize the probability as follows:

$$\arg\max_{\Theta} \sum_{v \in V} \sum_{c_t \in C_t(v)} \log p(v|c_t; \Theta). \tag{6.27}$$

To achieve efficient optimization, HHNE leverages the negative sampling proposed in [21], which basically samples a small number of negative objects to enhance the influence of positive objects. For a given node v, HHNE aims to maximize the proximity between v and its neighborhood c_t while minimizes the proximity between v and its negative sampled node n. Therefore, the objective function Eq. (6.27) can be rewritten as follows:

$$\mathcal{L}(\Theta) = \log\sigma[-d_{\mathbb{D}}(\theta_{c_t}, \theta_v)] + \sum_{m=1}^{M} \mathbb{E}_{n^m \sim P(n)}\{\log\sigma[d_{\mathbb{D}}(\theta_{n^m}, \theta_v)]\}, \tag{6.28}$$

where $P(n)$ is a pre-defined distribution from which a negative node n^m is drew from for M times. HHNE builds the node frequency distribution by drawing nodes regardless of their types.

6.4.2.3 Optimization

As the parameters of the model live in a Poincaré ball which has a Riemannian manifold structure, the back-propagated gradient is a Riemannian gradient. It means that the Euclidean gradient based optimization, such as $\theta_i \leftarrow \theta_i + \eta\nabla^E_{\theta_i}\mathcal{L}(\Theta)$, makes no sense as an operation in the Poincaré ball, because the addition operation is not defined in this manifold. Instead, HHNE can optimize Eq. (6.28) via a Riemannian stochastic gradient descent (RSGD) optimization method [1]. In particular, let $\mathcal{T}_{\theta_i}\mathbb{D}^d$ denote the tangent space of a node embedding $\theta_i \in \mathbb{D}^d$, and HHNE can compute the Riemannian gradient $\nabla^R_{\theta_i}\mathcal{L}(\Theta) \in \mathcal{T}_{\theta_i}\mathbb{D}^d$ of $\mathcal{L}(\Theta)$. Using RSGD, HHNE can be optimized by maximizing Eq. (6.28), and a node embedding can be updated in the form of:

$$\theta_i \leftarrow \exp_{\theta_i}(\eta\nabla^R_{\theta_i}\mathcal{L}(\Theta)), \tag{6.29}$$

where $\exp_{\theta_i}(\cdot)$ is exponential map in the Poincaré ball. The exponential map is given by [11]:

$$\begin{aligned}
&\exp_{\theta_i}(s) = \\
&\frac{\lambda_{\theta_i}\left(\cosh(\lambda_{\theta_i}\|s\|) + \langle\theta_i, \frac{s}{\|s\|}\rangle\sinh(\lambda_{\theta_i}\|s\|)\right)}{1 + (\lambda_{\theta_i} - 1)\cosh(\lambda_{\theta_i}\|s\|) + \lambda_{\theta_i}\langle\theta_i, \frac{s}{\|s\|}\rangle\sinh(\lambda_{\theta_i}\|s\|)}\theta_i \\
&+ \frac{\frac{1}{\|s\|}\sinh(\lambda_{\theta_i}\|s\|)}{1 + (\lambda_{\theta_i} - 1)\cosh(\lambda_{\theta_i}\|s\|) + \lambda_{\theta_i}\langle\theta_i, \frac{s}{\|s\|}\rangle\sinh(\lambda_{\theta_i}\|s\|)}s.
\end{aligned} \tag{6.30}$$

As the Poincaré ball model is a conformal model of hyperbolic spaces, i.e., $g^{\mathbb{D}}_x = \lambda^2_x g^{\mathbb{E}}$, the Riemannian gradient ∇^R is obtained by rescaling the Euclidean gradient ∇^E by the inverse of the metric tensor, i.e., $\frac{1}{g^{\mathbb{D}}_x}$:

$$\nabla^R_{\theta_i}\mathcal{L} = \left(\frac{1}{\lambda_{\theta_i}}\right)^2\nabla^E_{\theta_i}\mathcal{L}. \tag{6.31}$$

Furthermore, the gradients of Eq. (6.28) can be derived as follows:

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial \theta_{u^m}} =& \frac{4}{\alpha\sqrt{\gamma^2-1}}\left[\mathbb{I}_v[u^m] - \sigma(-d_{\mathbb{D}}(\theta_{c_t}, \theta_{u^m}))\right] \\ & \cdot \left[\frac{\theta_{c_t}}{\beta_m} - \frac{\|\theta_{c_t}\|^2 - 2\langle\theta_{c_t}, \theta_{u^m}\rangle + 1}{\beta_m^2}\theta_{u^m}\right],\end{aligned} \tag{6.32}$$

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial \theta_{c_t}} =& \sum_{m=0}^{M} \frac{4}{\beta_m\sqrt{\gamma^2-1}}\left[\mathbb{I}_v[u^m] - \sigma(-d_{\mathbb{D}}(\theta_{c_t}, \theta_{u^m}))\right] \\ & \cdot \left[\frac{\theta_{u^m}}{\alpha} - \frac{\|\theta_{u^m}\|^2 - 2\langle\theta_{c_t}, \theta_{u^m}\rangle + 1}{\alpha^2}\theta_{c_t}\right],\end{aligned} \tag{6.33}$$

where $\alpha = 1 - \|\theta_{c_t}\|^2$, $\beta_m = 1 - \|\theta_{u^m}\|^2$, $\gamma = 1 + \frac{2}{\alpha\beta}\|\theta_{c_t} - \theta_{u^m}\|^2$ and when $m = 0$, $u^0 = v$. $\mathbb{I}_v[u]$ is an indicator function to indicate whether u is v. Then, HHNE can be updated by using Eq. (6.32)-(6.33) iteratively.

6.4.3 Experiments

6.4.3.1 Experimental Setup

Datasets. The basic statistics of the two HGs used in our experiments are shown in Table 6.6.

Table 6.6 Statistics of datasets.

DBLP	# A	# P	# V	# P-A	# P-V
	14475	14376	20	41794	14376
MovieLens	# A	# M	# D	# M-A	# M-D
	11718	9160	3510	64051	9160

Baselines. HHNE is compared with the following state-of-the-art methods: (1) the homogeneous graph embedding methods, i.e., DeepWalk [25], LINE [28] and node2vec [13]; (2) the heterogeneous graph embedding methods, i.e., metapath2vec [8]; (3) the hyperbolic homogeneous graph embedding methods, i.e., PoincaréEmb [22].

Parameter Settings. For random walk based methods DeepWalk, node2vec, metapath2vec and HHNE, we set neighborhood size as 5, walk length as 80, walks per node as 40. For LINE, metapath2vec, PoincaréEmb and HHNE, we set the number of negative samples as 10. For methods based on meta-path guided random walks, we use "APA" for relation "P-A" in network reconstruction and link prediction experiments in DBLP; "APVPA" for relation "P-V" in above experiments in DBLP; "AMDMA"

for all relation in above experiments in MovieLens. In visualization experiment, in order to focus on analyzing the relation of "A" and "P", we use "APA".

6.4.3.2 Network Reconstruction

Table 6.7 AUC scores for network reconstruction.

Dataset	Edge	Dimension	Deepwalk	LINE(1st)	LINE(2nd)	node2vec	metapath2vec	PoincaréEmb	HHNE
DBLP	P-A	2	0.6933	0.5286	0.6740	0.7107	0.6686	0.8251	**0.9835**
		5	0.8034	0.5397	0.7379	0.8162	0.8261	0.8769	**0.9838**
		10	0.9324	0.6740	0.7541	0.9418	0.9202	0.8921	**0.9887**
		15	0.9666	0.7220	0.7868	0.9719	0.9500	0.8989	**0.9898**
		20	0.9722	0.7457	0.7600	0.9809	0.9623	0.9024	**0.9913**
		25	0.9794	0.7668	0.7621	0.9881	0.9690	0.9034	**0.9930**
	P-V	2	0.7324	0.5182	0.6242	0.7595	0.7286	0.5718	**0.8449**
		5	0.7906	0.5500	0.6349	0.8019	0.9072	0.5529	**0.9984**
		10	0.8813	0.7070	0.6333	0.8922	0.9691	0.6271	**0.9985**
		15	0.9353	0.7295	0.6343	0.9382	0.9840	0.6446	**0.9985**
		20	0.9505	0.7369	0.6444	0.9524	0.9879	0.6600	**0.9985**
		25	0.9558	0.7436	0.6440	0.9596	0.9899	0.6760	**0.9985**
MoiveLens	M-A	2	0.6320	0.5424	0.6378	0.6402	0.6404	0.5231	**0.8832**
		5	0.6763	0.5675	0.7047	0.6774	0.6578	0.5317	**0.9168**
		10	0.7610	0.6202	0.7739	0.7653	0.7231	0.5404	**0.9211**
		15	0.8244	0.6593	0.7955	0.8304	0.7793	0.5479	**0.9221**
		20	0.8666	0.6925	0.8065	0.8742	0.8189	0.5522	**0.9239**
		25	0.8963	0.7251	0.8123	0.9035	0.8483	0.5545	**0.9233**
	M-D	2	0.6626	0.5386	0.6016	0.6707	0.6589	0.6213	**0.9952**
		5	0.7263	0.5839	0.6521	0.7283	0.7230	0.7266	**0.9968**
		10	0.8246	0.6114	0.6969	0.8308	0.8063	0.7397	**0.9975**
		15	0.8784	0.6421	0.7112	0.8867	0.8455	0.7378	**0.9972**
		20	0.9117	0.6748	0.7503	0.9186	0.8656	0.7423	**0.9982**
		25	0.9345	0.7012	0.7642	0.9402	0.8800	0.7437	**0.9992**

A good HG embedding method should ensure that the learned embeddings can preserve the original HG structure. The reconstruction error in relation to the embedding dimension is then a measure for the capacity of the model. More specifically, we use network embedding methods to learn feature representations. Then for each type of links in the HG, we enumerate all pairs of objects that can be connected by such a link and calculate their proximity [18], i.e., the distance in Poincaré ball model for HHNE and PoincaréEmb. Finally, we use the AUC [9] to evaluate the performance of each embedding method. For example, for link type "write", we calculate all pairs of authors and papers in DBLP and compute the proximity for each pair. Then using the links between authors and papers in real DBLP network as ground-truth, we compute the AUC value for each embedding method.

The results are shown in Table 6.7. As we can see, HHNE consistently performs the best in all the tested HGs. The results demonstrate that HHNE can effectively preserve the original network structure and reconstruct the network, especially on the reconstruction of P-V and M-D edges. Also, please note that HHNE achieves very promising results when the embedding dimension is very small. This suggests that regarding hyperbolic spaces underlying HG is reasonable and hyperbolic spaces have strong ability of modeling network when the dimension of spaces is small.

Table 6.8 AUC scores for link prediction.

Dataset	Edge	Dimension	Deepwalk	LINE(1st)	LINE(2nd)	node2vec	metapath2vec	PoincaréEmb	HHNE
DBLP	P-A	2	0.5813	0.5090	0.5909	0.6709	0.6536	0.6742	**0.8777**
		5	0.7370	0.5168	0.6351	0.7527	0.7294	0.7381	**0.9041**
		10	0.8250	0.5427	0.6510	0.8469	0.8279	0.7699	**0.9111**
		15	0.8664	0.5631	0.6582	0.8881	0.8606	0.7743	**0.9111**
		20	0.8807	0.5742	0.6644	0.9037	0.8740	0.7806	**0.9106**
		25	0.8878	0.5857	0.6782	0.9102	0.8803	0.7830	**0.9117**
	P-V	2	0.7075	0.5160	0.5121	0.7369	0.7059	0.8257	**0.9331**
		5	0.7197	0.5663	0.5216	0.7286	0.8516	0.8878	**0.9409**
		10	0.7292	0.5873	0.5332	0.7481	0.9248	0.9113	**0.9619**
		15	0.7325	0.5896	0.5425	0.7583	0.9414	0.9142	**0.9625**
		20	0.7522	0.5891	0.5492	0.7674	0.9504	0.9185	**0.9620**
		25	0.7640	0.5846	0.5512	0.7758	0.9536	0.9192	**0.9612**
MoiveLens	M-A	2	0.6278	0.5053	0.5712	0.6349	0.6168	0.5535	**0.7715**
		5	0.6353	0.5636	0.5874	0.6402	0.6212	0.5779	**0.8255**
		10	0.6680	0.5914	0.6361	0.6700	0.6332	0.5984	**0.8312**
		15	0.6791	0.6184	0.6442	0.6814	0.6382	0.5916	**0.8319**
		20	0.6868	0.6202	0.6596	0.6910	0.6453	0.5988	**0.8318**
		25	0.6890	0.6256	0.6700	0.6977	0.6508	0.5995	**0.8309**
	M-D	2	0.6258	0.5139	0.6501	0.6299	0.6191	0.5856	**0.8520**
		5	0.6482	0.5496	0.6607	0.6589	0.6332	0.6290	**0.8967**
		10	0.6976	0.5885	0.7499	0.7034	0.6687	0.6518	**0.8984**
		15	0.7163	0.6647	0.7756	0.7241	0.6702	0.6715	**0.9007**
		20	0.7324	0.6742	0.7982	0.7412	0.6746	0.6821	**0.9000**
		25	0.7446	0.6957	0.8051	0.7523	0.6712	0.6864	**0.9018**

6.4.3.3 Link Prediction

Link prediction aims to infer the unknown links in an HG given the observed HG structure, which can be used to test the generalization performance of a network embedding method. The experimental setting is similar to [34]. For each type of edge, 20% of edges are removed randomly from the network while ensuring that the rest network structure is still connected. The proximity of all pair of nodes are calculated in the test. AUC is used as the evaluation metric.

From the results in Table 6.8, HHNE outperforms the baselines upon all the dimensionality, especially in the low dimensionality. The results can demonstrate the generalization ability of HHNE. In DBLP dataset, the results of HHNE in 10 dimensionality exceed all the baselines in higher dimensionality results. In MovieLens dataset, HHNE with only 2 dimensionality surpasses baselines in all dimensionality. Besides, both of LINE(1st) and PoincaréEmb preserve proximities of node pairs linked by an edge, while LINE(1st) embed network into Euclidean spaces and Poincaré embed network into hyperbolic spaces. PoincaréEmb performs better than LINE(1st) in most cases, especially in dimensionality lower than 10, suggesting the superiority of embedding network into hyperbolic spaces. Because HHNE can preserve high-order network structure and handle different types of nodes in HG, HHNE is more effective than PoincaréEmb.

More detailed introduction of HHNE can be found in [33].

6.5 Conclusion

In this chapter, we have introduced three emerging HG embedding topics, as well as the related HG embedding methods. For adversarial learning, HeGAN is designed for learning semantic-preserving and robust HG embedding based on the adversarial principle. Besides, for importance sampling, HeteSamp studies the problem of accelerating large-scale HG embedding with importance sampling. Moreover, for hyperbolic representation, HHNE makes the efforts to embed HGs in hyperbolic spaces, and the optimization strategies are derived to optimize the hyperbolic embedding. We hope more HG embedding methods with deeper insights can be proposed to discover the abundant semantic of HG in the future.

References

1. Bonnabel, S., et al.: Stochastic gradient descent on riemannian manifolds. IEEE Trans. Automat. Contr. **58**(9), 2217–2229 (2013)
2. Cai, X., Han, J., Yang, L.: Generative adversarial network based heterogeneous bibliographic network representation for personalized citation recommendation. In: AAAI, pp. 5747–5754 (2018)
3. Cannon, J.W., Floyd, W.J., Kenyon, R., Parry, W.R., et al.: Hyperbolic geometry. Flavors of geometry **31**, 59–115 (1997)
4. Cen, Y., Zou, X., Zhang, J., Yang, H., Zhou, J., Tang, J.: Representation learning for attributed multiplex heterogeneous network. In: KDD, pp. 1358–1368 (2019)
5. Chen, J., Ma, T., Xiao, C.: Fastgcn: Fast learning with graph convolutional networks via importance sampling. In: ICLR (2018)
6. Dai, Q., Li, Q., Tang, J., Wang, D.: Adversarial network embedding. In: AAAI, pp. 2167–2174 (2018)
7. Dhingra, B., Shallue, C., Norouzi, M., Dai, A., Dahl, G.: Embedding text in hyperbolic spaces. In: TextGraphs, pp. 59–69 (2018)
8. Dong, Y., Chawla, N.V., Swami, A.: metapath2vec: Scalable representation learning for heterogeneous networks. In: KDD, pp. 135–144 (2017)
9. Fawcett, T.: An introduction to roc analysis. Pattern recognition letters **27**(8), 861–874 (2006)
10. Fu, T.y., Lee, W.C., Lei, Z.: Hin2vec: Explore meta-paths in heterogeneous information networks for representation learning. In: CIKM, pp. 1797–1806 (2017)
11. Ganea, O., Becigneul, G., Hofmann, T.: Hyperbolic entailment cones for learning hierarchical embeddings. In: ICML, pp. 1646–1655 (2018)
12. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative adversarial nets. In: NeurIPS, pp. 2672–2680 (2014)
13. Grover, A., Leskovec, J.: node2vec: Scalable feature learning for networks. In: KDD, pp. 855–864 (2016)
14. Hamilton, W.L., Ying, Z., Leskovec, J.: Inductive representation learning on large graphs. In: NeurIPS, pp. 1025–1035 (2017)
15. He, X., Liao, L., Zhang, H., Nie, L., Hu, X., Chua, T.S.: Neural collaborative filtering. In: WWW, pp. 173–182 (2017)
16. Hu, B., Fang, Y., Shi, C.: Adversarial learning on heterogeneous information networks. In: A. Teredesai, V. Kumar, Y. Li, R. Rosales, E. Terzi, G. Karypis (eds.) KDD, pp. 120–129 (2019)
17. Huang, W., Zhang, T., Rong, Y., Huang, J.: Adaptive sampling towards fast graph representation learning. In: NeurIPS, pp. 4563–4572 (2018)

18. Huang, Z., Mamoulis, N.: Heterogeneous information network embedding for meta path based proximity. arXiv preprint arXiv:1701.05291 (2017)
19. Ji, Y., Yin, M., Yang, H., Zhou, J., Zheng, V.W., Shi, C., Fang, Y.: Accelerating large-scale heterogeneous interaction graph embedding learning via importance sampling. ACM Transactions on Knowledge Discovery from Data **15**(1), 1–23 (2020)
20. Krioukov, D., Papadopoulos, F., Kitsak, M., Vahdat, A., Boguná, M.: Hyperbolic geometry of complex networks. Physical Review E **82**(3), 036,106 (2010)
21. Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S., Dean, J.: Distributed representations of words and phrases and their compositionality. In: NIPS, pp. 3111–3119 (2013)
22. Nickel, M., Kiela, D.: Poincaré embeddings for learning hierarchical representations. In: NeurIPS, pp. 6338–6347 (2017)
23. Owen, A.B.: Monte Carlo theory, methods and examples (2013)
24. Pan, S., Hu, R., Long, G., Jiang, J., Yao, L., Zhang, C.: Adversarially regularized graph autoencoder for graph embedding. In: IJCAI, pp. 2609–2615 (2018)
25. Perozzi, B., Al-Rfou, R., Skiena, S.: Deepwalk: Online learning of social representations. In: KDD, pp. 701–710 (2014)
26. Reed, S., Akata, Z., Yan, X., Logeswaran, L., Schiele, B., Lee, H.: Generative adversarial text to image synthesis. In: ICML, pp. 1060–1069 (2016)
27. Shi, C., Hu, B., Zhao, X., Yu, P.: Heterogeneous information network embedding for recommendation. IEEE Transactions on Knowledge and Data Engineering **31**(2), 357–370 (2018)
28. Tang, J., Qu, M., Wang, M., Zhang, M., Yan, J., Mei, Q.: Line: Large-scale information network embedding. In: WWW, pp. 1067–1077 (2015)
29. Wang, H., Wang, J., Wang, J., Zhao, M., Zhang, W., Zhang, F., Xie, X., Guo, M.: Graphgan: Graph representation learning with generative adversarial nets. In: AAAI, pp. 2508–2515 (2018)
30. Wang, H., Wang, J., Wang, J., Zhao, M., Zhang, W., Zhang, F., Xie, X., Guo, M.: Graphgan: Graph representation learning with generative adversarial nets. In: AAAI, pp. 2508–2515 (2018)
31. Wang, J., Yu, L., Zhang, W., Gong, Y., Xu, Y., Wang, B., Zhang, P., Zhang, D.: Irgan: A minimax game for unifying generative and discriminative information retrieval models. In: SIGIR, pp. 515–524 (2017)
32. Wang, X., Ji, H., Shi, C., Wang, B., Ye, Y., Cui, P., Yu, P.S.: Heterogeneous graph attention network. In: WWW, pp. 2022–2032 (2019)
33. Wang, X., Zhang, Y., Shi, C.: Hyperbolic heterogeneous information network embedding. In: AAAI, pp. 5337–5344 (2019)
34. Xu, L., Wei, X., Cao, J., Yu, P.S.: Embedding of embedding (eoe): Joint embedding for coupled heterogeneous networks. In: WSDM, pp. 741–749 (2017)
35. Ying, R., He, R., Chen, K., Eksombatchai, P., Hamilton, W.L., Leskovec, J.: Graph convolutional neural networks for web-scale recommender systems. In: KDD, pp. 974–983 (2018)
36. Yu, L., Zhang, W., Wang, J., Yu, Y.: Seqgan: Sequence generative adversarial nets with policy gradient. In: AAAI, pp. 2852–2858 (2017)
37. Yu, W., Zheng, C., Cheng, W., Aggarwal, C.C., Song, D., Zong, B., Chen, H., Wang, W.: Learning deep network representations with adversarially regularized autoencoders. In: KDD, pp. 2663–2671 (2018)
38. Zheng, V.W., Sha, M., Li, Y., Yang, H., Fang, Y., Zhang, Z., Tan, K., Chang, K.C.: Heterogeneous embedding propagation for large-scale e-commerce user alignment. In: ICDM, pp. 1434–1439 (2018)
39. Zou, D., Hu, Z., Wang, Y., Jiang, S., Sun, Y., Gu, Q.: Layer-dependent importance sampling for training deep and large graph convolutional networks. In: NeurIPS, pp. 11,247–11,256 (2019)