Heterogeneous Graph Neural Network

Chuxu Zhang
University of Notre Dame
czhang11@nd.edu

Dongjin Song
NEC Laboratories America, Inc.
dsong@nec-labs.com

Chao Huang
University of Notre Dame, JD Digits
chuang7@nd.edu

Ananthram Swami
US Army Research Laboratory
ananthram.swami.civ@mail.mil

Nitesh V. Chawla
University of Notre Dame
nchawla@nd.edu

ABSTRACT

Representation learning in heterogeneous graphs aims to pursue a meaningful vector representation for each node so as to facilitate downstream applications such as link prediction, personalized recommendation, node classification, *etc*. This task, however, is challenging not only because of the demand to incorporate heterogeneous structural (graph) information consisting of multiple types of nodes and edges, but also due to the need for considering heterogeneous attributes or contents (*e.g.*, text or image) associated with each node. Despite a substantial amount of effort has been made to homogeneous (or heterogeneous) graph embedding, attributed graph embedding as well as graph neural networks, few of them can jointly consider heterogeneous structural (graph) information as well as heterogeneous contents information of each node effectively. In this paper, we propose HetGNN, a heterogeneous graph neural network model, to resolve this issue. Specifically, we first introduce a random walk with restart strategy to sample a fixed size of strongly correlated heterogeneous neighbors for each node and group them based upon node types. Next, we design a neural network architecture with two modules to aggregate feature information of those sampled neighboring nodes. The first module encodes "deep" feature interactions of heterogeneous contents and generates content embedding for each node. The second module aggregates content (attribute) embeddings of different neighboring groups (types) and further combines them by considering the impacts of different groups to obtain the ultimate node embedding. Finally, we leverage a graph context loss and a mini-batch gradient descent procedure to train the model in an end-to-end manner. Extensive experiments on several datasets demonstrate that HetGNN can outperform state-of-the-art baselines in various graph mining tasks, *i.e.*, link prediction, recommendation, node classification & clustering and inductive node classification & clustering.

KEYWORDS

Heterogeneous graphs, Graph neural networks, Graph embedding

ACM Reference Format:
Chuxu Zhang, Dongjin Song, Chao Huang, Ananthram Swami, and Nitesh V. Chawla. 2019. Heterogeneous Graph Neural Network. In *The 25th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '19), August 4–8, 2019, Anchorage, AK, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3292500.3330961

KDD '19, August 4–8, 2019, Anchorage, AK, USA

ACM ISBN 978-1-4503-6201-6/19/08…$15.00
https://doi.org/10.1145/3292500.3330961

1 INTRODUCTION

Heterogeneous graphs (HetG) [26, 27] contain abundant information with structural relations (edges) among multi-typed nodes as well as unstructured content associated with each node. For instance, the academic graph in Fig. 1(a) denotes relations between authors and papers (write), papers and papers (cite), papers and venues (publish), *etc*. Moreover, nodes in this graph carry attributes (*e.g.*, author id) and text (*e.g.*, paper abstract). Another example illustrates user-item relations in the review graph and nodes are associated with attributes (*e.g.*, user id), text (*e.g.*, item description) and image (*e.g.*, item picture). This ubiquity of HetG has led to an influx of research on corresponding graph mining methods and algorithms such as relation inference [2, 25, 33, 35], personalized recommendation [10, 23], node classification [36], *etc*.

Traditionally, a variety of these HetG tasks have relied on feature vectors derived from a manual feature engineering tasks. This requires specifications and computation of different statistics or properties about the HetG as a feature vector for downstream machine learning or analytic tasks. However, this can be very limiting and not generalizable. More recently, there has been an emergence of representation learning approaches to automate the feature engineering tasks, which can then facilitate a multitude of downstream machine learning or analytic tasks. Beginning with homogeneous graphs [6, 20, 29], graph representation learning has been expanded to heterogeneous graphs [1, 4], attributed graphs [15, 34] as well as specific graphs [22, 28]. For instance, the "shallow" models, *e.g.*, DeepWalk [20], were initially developed to feed a set of short random walks over the graph to the SkipGram model [19] so as to approximate the node co-occurrence probability in these walks and obtain node embeddings. Subsequently, semantic-aware approaches, *e.g.*, metapath2vec [4], were proposed to address node and relation heterogeneity in heterogeneous graphs. In addition, content-aware approaches, *e.g.*, ASNE [15], leveraged both "latent" features and attributes to learn node embeddings in the graph.

These methods learn node "latent" embeddings directly, but are limited in capturing the rich neighborhood information. The Graph Neural Networks (GNNs) employ deep neural networks to aggregate feature information of neighboring nodes, which makes the aggregated embedding more powerful. In addition, the GNNs can be naturally applied to inductive tasks involving nodes that are not present in the training period. For instance, GCN [12], GraphSAGE

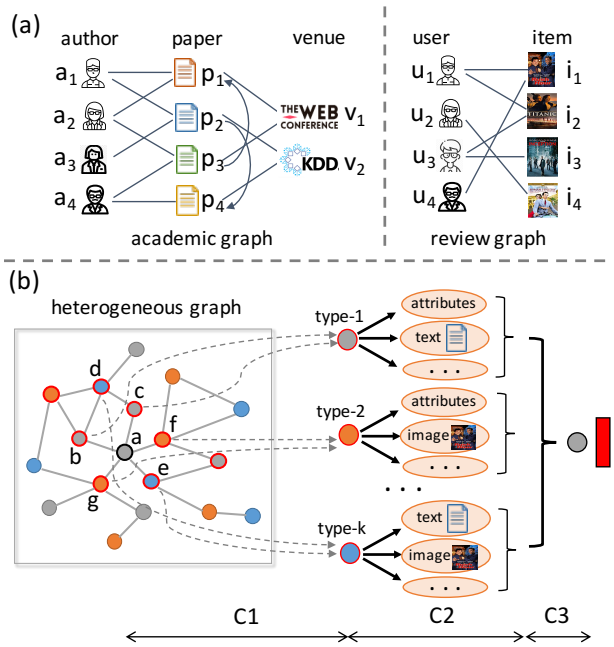

Figure 1: (a) HetG examples: an academic graph and a review graph. (b) Challenges of graph neural network for HetG: C1 - sampling heterogeneous neighbors (for node *a* in this case, node colors denote different types); C2 - encoding heterogeneous contents; C3 - aggregating heterogeneous neighbors.

[7], and GAT [31] employ convolutional operator, LSTM architecture, and self-attention mechanism to aggregate feature information of neighboring nodes, respectively. The advances and applications of GNNs are largely concentrated on homogeneous graphs. Current state-of-the-art GNNs have not well solved the following challenges faced for HetG, which we address in this paper.

- (**C1**) Many nodes in HetG may not connect to all types of neighbors. In addition, the number of neighboring nodes varies from node to node. For example, in Figure 1(a), any author node has no direct connection to a venue node. Meanwhile, in Figure 1(b), node *a* has 5 direct neighbors while node *c* only has 2. Most existing GNNs only aggregate feature information of direct (first-order) neighboring nodes and the feature propagation process may weaken the effect of farther neighbors. Moreover, the embedding generation of "hub" node is impaired by weakly correlated neighbors ("noise" neighbors) and the embedding of "cold-start" node is not sufficiently represented due to limited neighbor information. Thus challenge 1 is: *how to sample heterogeneous neighbors that are strongly correlated to embedding generation for each node in HetG, as indicated by C1 in Figure 1(b)?*
- (**C2**) A node in HetG can carry unstructured heterogeneous contents, *e.g.*, attributes, text or image. In addition, content associated with different types of nodes can be different. For example, in Figure 1(b), type-1 nodes (*e.g.*, *b* or *c*) contain attributes and text content, type-2 nodes (*e.g.*, *f* or *g*) carry attributes and image, type-k nodes (*e.g.*, *d* or *e*) are associated with text and image. The direct concatenation operation or linear transformation by the current GNNs cannot model "deep" interactions among node heterogeneous contents. Moreover, it is not applicable to use the same feature transformation function for all node types as their contents vary from each other. Thus challenge 2 is: *how to design node content encoder for addressing content heterogeneity of different nodes in HetG, as indicated by C2 in Figure 1(b)?*
- (**C3**) Different types of neighbors contribute differently to the node embeddings in HetG. For example, in the academic graph of Figure 1(a), author and paper neighbors should have more influence on the embedding of author node as a venue node contains diverse topics thus has more general embedding. Most of current GNNs mainly focus on homogeneous graphs and do not consider node type impact. Thus challenge 3 is: *how to aggregate feature information of heterogeneous neighbors by considering the impacts of different node types, as indicated by C3 in Figure 1(b).*

Table 1: Model comparison: (1) RL - representation learning? (2) HG - heterogeneous graph? (3) C - content aware? (4) HC - heterogeneous contents aware? (5) I - inductive inference?

Property	DW [20]	MP2V [4]	ASNE [15]	SHNE [34]	GSAGE [7]	GAT [31]	HetGNN
RL	✓	✓	✓	✓	✓	✓	✓
HG	✗	✓	✗	✓	✗	✗	✓
C	✗	✗	✓	✓	✓	✓	✓
HC	✗	✗	✓	✗	✓	✓	✓
I	✗	✗	✗	✗	✓	✓	✓

To solve these challenges, we propose HetGNN, a heterogeneous graph neural network model for representation learning in HetG. First, we design a random walk with restart based strategy to sample fixed size strongly correlated heterogeneous neighbors of each node in HetG and group them according to node types. Next, we design a heterogeneous graph neural network architecture with two modules to aggregate feature information of sampled neighbors in previous step. The first module employs recurrent neural network to encode "deep" feature interactions of heterogeneous contents and obtains content embedding of each node. The second module utilizes another recurrent neural network to aggregate content embeddings of different neighboring groups, which are further combined by an attention mechanism for measuring the different impacts of heterogeneous node types and obtaining the ultimate node embedding. Finally, we leverage a graph context loss and a mini-batch gradient descent procedure to train the model. To summarize, the main contributions of our work are:

- We formalize the problem of heterogeneous graph representation learning which involves both graph structure heterogeneity and node content heterogeneity.
- We propose an innovative heterogeneous graph neural network model, *i.e.*, HetGNN, for representation learning on HetG. HetGNN is able to capture both structure and content heterogeneity and is useful for both transductive and inductive tasks. Table 1 summarizes the key advantages of HetGNN, comparing to a number of recent models which include homogeneous, heterogeneous, attributed graph models, and graph neural network models.
- We conduct extensive experiments on several public datasets and our results demonstrate the superior performance of HetGNN over state-of-the-art baselines for numerous graph mining tasks

including link prediction, recommendation, node classification & clustering, and inductive node classification & clustering.

2 PROBLEM DEFINITION

In this section, we introduce the concept of content-associated heterogeneous graphs that will be used in the paper and then formally define the problem of heterogeneous graph representation learning.

Definition 2.1. Content-associated Heterogeneous Graphs. A content associated heterogeneous graph (C-HetG) is defined as a graph $G = (V, E, O_V, R_E)$ with multiple types of nodes V and links E. O_V and R_E represent the set of object types and that of relation types, respectively. In addition, each node is associated with heterogeneous contents, *e.g.*, attributes, text, or image.

The academic graph in Figure 1(a) is a C-HetG. The node types O_V includes *author*, *paper* and *venue*. The link types R_E includes *author-write-paper*, *paper-cite-paper* and *paper-publish-venue*. Besides, the author or venue node is associated with paper abstract written by the author or included in the venue, and the paper node contains abstract, references, as well as venue. The bipartite review graph in Figure 1(a) is also C-HetG as $|O_V| + |R_E| \geq 3$, where O_V includes user and item, the relation R_E indicates review behavior. The user node is associated with review that is written by the user and the item node contains title, description, and picture.

Problem 1. Heterogeneous Graph Representation Learning. Given a C-HetG $G = (V, E, O_V, R_E)$ with node content set C, the task is to design a model $\mathcal{F}_\Theta$ with parameters Θ to learn d-dimensional embeddings $\mathcal{E} \in \mathbb{R}^{|V| \times d} (d \ll |V|)$ that are able to encode both heterogeneous structural closeness and heterogeneous unstructured contents among them. The node embeddings can be utilized in various graph mining tasks, such as link prediction, recommendation, multi-labels classification, and node clustering.

3 HetGNN

In this section, we formally present HetGNN to resolve those three challenges described in Section 1. HetGNN consists of four parts: (1) sampling heterogeneous neighbors; (2) encoding node heterogeneous contents; (3) aggregating heterogeneous neighbors; (4) formulating the objective and designing model training procedure. Figure 2 illustrates the framework of HetGNN.

3.1 Sampling Heterogeneous Neighbors (C1)

The key idea of most graph neural networks (GNNs) is to aggregate feature information from a node's direct (first-order) neighbors, such as GraphSAGE [7] or GAT [31]. However, directly applying these approaches to heterogeneous graphs may raise several issues:

- They cannot directly capture feature information from different types of neighbors. For example, authors do not directly connect to local authors and venue neighbors in Fig. 1(a), which could lead to insufficient representation.
- They are weakened by various neighbor sizes. Some author writes many papers while some only have few papers in the academic graph. Some items are reviewed by many users while some receive few feedbacks in the review graph. The embedding of "hub" node could be impaired by weakly correlated neighbors and "cold-start" node embedding may not be sufficiently represented.
- They are not suitable for aggregating heterogeneous neighbors which have different content features. Heterogeneous neighbors may require different feature transformations to deal with different feature types and dimensions.

In light of these issues and to solve the challenge **C1**, we design a heterogeneous neighbors sampling strategy based on random walk with restart (RWR). It contains two consecutive steps:

- Step-1: Sampling fixed length RWR. We start a random walk from node $v \in V$. The walk iteratively travels to the neighbors of current node or returns to the starting node with a probability p. RWR runs until it successfully collects a fixed number of nodes, denoted as RWR(v). Note that numbers of different types of nodes in RWR(v) are constrained to ensure that all node types are sampled for v.
- Step-2: Grouping different types of neighbors. For each node type t, we select top k_t nodes from RWR(v) according to frequency and take them as the set of t-type correlated neighbors of node v.

This strategy is able to avoid the aforementioned issues due to: (1) RWR collects all types of neighbors for each node; (2) the sampled neighbor size of each node is fixed and the most frequently visited neighbors are selected; (3) neighbors of the same type (having the same content features) are grouped such that type-based aggregation can be deployed. Next, we design a heterogeneous graph neural network architecture with two modules to aggregate feature information of the sampled heterogeneous neighbors for each node.

3.2 Encoding Heterogeneous Contents (C2)

To solve the challenge **C2**, we design a module to extract heterogeneous contents C_v from node $v \in V$ and encode them as a fixed size embedding via a neural network f_1. Specifically, we denote the feature representation of i-th content in C_v as $\mathbf{x}_i \in \mathbb{R}^{d_f \times 1}$ (d_f: content feature dimension). Note that $\mathbf{x}_i$ can be pre-trained using different techniques *w.r.t.* different types of contents. For example, we can utilize Par2Vec [13] to pre-train text content or employ CNNs [17] to pre-train image content. Unlike the previous models [7, 31] that concatenate different content features directly or linearly transform them into an unified vector, we design a new architecture based on bi-directional LSTM (Bi-LSTM) [9] to capture "deep" feature interactions and obtain larger expressive capability. Formally, the content embedding of v is computed as follows:

$$f_1(v) = \frac{\sum_{i \in C_v} \left[\overrightarrow{LSTM} \left\{ \mathcal{FC}_{\theta_x}(\mathbf{x}_i) \right\} \bigoplus \overleftarrow{LSTM} \left\{ \mathcal{FC}_{\theta_x}(\mathbf{x}_i) \right\} \right]}{|C_v|} \tag{1}$$

where $f_1(v) \in \mathbb{R}^{d \times 1}$ (d: content embedding dimension), $\mathcal{FC}_{\theta_x}$ denotes feature transformer which can be identity (no transformation), fully connected neural network with parameter θ_x, *etc*. The operator $\bigoplus$ denotes concatenation. The LSTM is formulated as:

$$\begin{aligned}
\mathbf{z}_i &= \sigma(\mathcal{U}_z \mathcal{FC}_{\theta_x}(\mathbf{x}_i) + \mathcal{W}_z \mathbf{h}_{i-1} + \mathbf{b}_z) \\
\mathbf{f}_i &= \sigma(\mathcal{U}_f \mathcal{FC}_{\theta_x}(\mathbf{x}_i) + \mathcal{W}_f \mathbf{h}_{i-1} + \mathbf{b}_f) \\
\mathbf{o}_i &= \sigma(\mathcal{U}_o \mathcal{FC}_{\theta_x}(\mathbf{x}_i) + \mathcal{W}_o \mathbf{h}_{i-1} + \mathbf{b}_o) \\
\hat{\mathbf{c}}_i &= \tanh(\mathcal{U}_c \mathcal{FC}_{\theta_x}(\mathbf{x}_i) + \mathcal{W}_c \mathbf{h}_{i-1} + \mathbf{b}_c) \\
\mathbf{c}_i &= \mathbf{f}_i \circ \mathbf{c}_{i-1} + \mathbf{z}_i \circ \hat{\mathbf{c}}_i \\
\mathbf{h}_i &= \tanh(\mathbf{c}_i) \circ \mathbf{o}_i
\end{aligned} \tag{2}$$

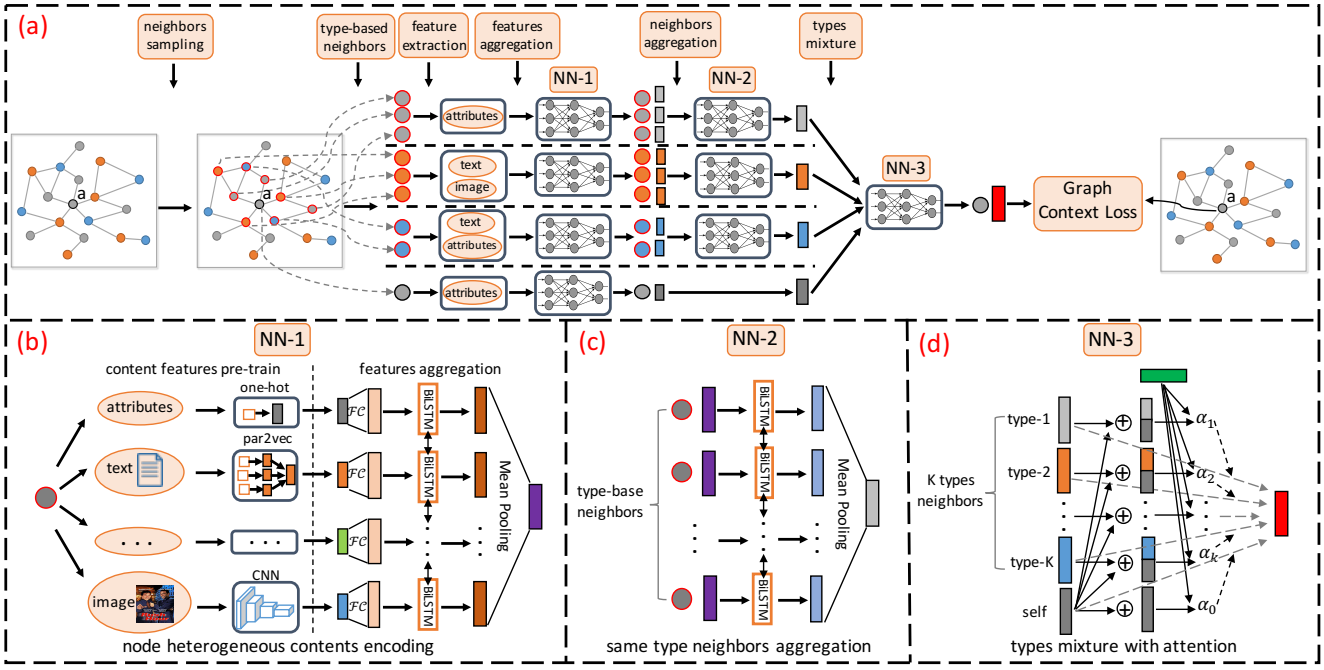

Figure 2: (a) The overall architecture of HetGNN: it first samples fix sized heterogeneous neighbors for each node (node a in this case), next encodes each node content embedding via NN-1, then aggregates content embeddings of the sampled heterogeneous neighbors through NN-2 and NN-3, finally optimizes the model via a graph context loss; (b) NN-1: node heterogeneous contents encoder; (c) NN-2: type-based neighbors aggregator; (d) NN-3: heterogeneous types combination.

where $\mathbf{h}_i \in \mathbb{R}^{(d/2)\times 1}$ is the output hidden state of i-th content, $\circ$ denotes Hadamard product, $\mathcal{U}_j \in \mathbb{R}^{(d/2)\times d_f}$, $\mathcal{W}_j \in \mathbb{R}^{(d/2)\times(d/2)}$, and $\mathbf{b}_j \in \mathbb{R}^{(d/2)\times 1}$ ($j \in \{z, f, o, c\}$) are learnable parameters, $\mathbf{z}_i$, $\mathbf{f}_i$, and $\mathbf{o}_i$ are forget gate vector, input gate vector, and output gate vector of i-th content feature, respectively. To be more specific, the above architecture first uses different $\mathcal{FC}$ layers to transform different content features, then employs the Bi-LSTM to capture "deep" feature interactions and accumulate expression capability of all content features, and finally utilizes a mean pooling layer over all hidden states to obtain the general content embedding of v, as illustrated in Figure 2(b). Note that the Bi-LSTM operates on an unordered content set C_v, which is inspired by previous work [7] for aggregating unordered neighbors. Besides, we use different Bi-LSTMs to aggregate content features for different types of nodes as their contents vary from each other.

There are three main advantages for this encoding architecture: (1) it has concise structures with relative low complexity (less parameters), making the model implementation and tuning relatively easy; (2) it is capable to fuse the heterogeneous contents information, leading to a strong expression capability; (3) it is flexible to add extra content features, making the model extension convenient.

3.3 Aggregating Heterogeneous Neighbors (C3)

To aggregate content embeddings (obtained from Section 3.2) of heterogeneous neighbors for each node and solve the challenge **C3**, we design another module which is a type-based neural network. It includes two consecutive steps: (1) same type neighbors aggregation; (2) types combination.

3.3.1 *Same Type Neighbors Aggregation.*

In Section 3.1, we use RWR based strategy to sample fixed size neighbor sets of different node types for each node. Accordingly, we denote the t-type sampled neighbor set of $v \in V$ as $N_t(v)$. Then, we employ a neural network f_2^t to aggregate content embeddings of $v' \in N_t(v)$. Formally, the aggregated t-type neighbors embedding for v is formulated as follows:

$$f_2^t(v) = \mathcal{AG}^t_{v' \in N_t(v)} \left\{ f_1(v') \right\} \tag{3}$$

where $f_2^t(v) \in \mathbb{R}^{d\times 1}$ (d: aggregated content embedding dimension), $f_1(v')$ is the content embedding of v' generated by the module in Section 3.2, $\mathcal{AG}^t$ is the t-type neighbors aggregator which can be fully connected neural network, convolutional neural network, recurrent neural network, *etc*. In this work, we use the Bi-LSTM since it yields better performance in practise. Thus we re-formulate $f_2^t(v)$ as follows:

$$f_2^t(v) = \frac{\sum_{v' \in N_t(v)} \left[\overrightarrow{LSTM} \left\{ f_1(v') \right\} \bigoplus \overleftarrow{LSTM} \left\{ f_1(v') \right\} \right]}{|N_t(v)|} \tag{4}$$

where LSTM module has the same formulation as Eq. (2) except input and parameter set. Obviously, we employ Bi-LSTM to aggregate content embeddings of all t-type neighbors and use the average over all hidden states to represent the general aggregated embedding, as illustrated in Figure 2(c). We use different Bi-LSTMs to distinguish different node types for neighbors aggregation. Note that the Bi-LSTM operates on an unordered neighbors set, which is inspired by GraphSAGE [7].

3.3.2 Types Combination.
The previous step generates $|O_V|$ (O_V: set of node types in the graph) aggregated embeddings for node v. To combine these type-based neighbor embeddings with v's content embedding, we employ the attention mechanism [31]. The motivation is that different types of neighbors will make different contributions to the final representation of v. Thus the output embedding is formulated as:

$$\mathcal{E}_v = \alpha^{v,v} f_1(v) + \sum_{t \in O_V} \alpha^{v,t} f_2^t(v) \tag{5}$$

where $\mathcal{E}_v \in \mathbb{R}^{d \times 1}$ (d: output embedding dimension), $\alpha^{v,*}$ indicates the importance of different embeddings, $f_1(v)$ is the content embedding of v obtained from Section 3.2, $f_2^t(v)$ is the type-based aggregated embedding obtained from Section 3.3. We denote the set of embeddings as $\mathcal{F}(v) = \{f_1(v) \bigcup (f_2^t(v), t \in O_V)\}$ and reformulate the output embedding of v as:

$$\begin{aligned} \mathcal{E}_v &= \sum_{f_i \in \mathcal{F}(v)} \alpha^{v,i} f_i \\ \alpha^{v,i} &= \frac{exp\left\{LeakyReLU(u^T[f_i \bigoplus f_1(v)])\right\}}{\sum_{f_j \in \mathcal{F}(v)} exp\left\{LeakyReLU(u^T[f_j \bigoplus f_1(v)])\right\}} \end{aligned} \tag{6}$$

where $LeakyReLU$ denotes leaky version of a Rectified Linear Unit, $u \in \mathbb{R}^{2d \times 1}$ is the attention parameter. Figure 2(c) gives the illustration of this step.

In this framework, to make embedding dimension consistent and model tuning easy, we use the same dimension d for content embedding in Section 3.2, aggregated content embedding in Section 3.3, and output node embedding in Section 3.3.

3.4 Objective and Model Training

To perform heterogeneous graph representation learning, we define the following objective with parameters Θ:

$$o_1 = \arg\max_{\Theta} \prod_{v \in V} \prod_{t \in O_V} \prod_{v_c \in CN_v^t} p(v_c|v;\Theta) \tag{7}$$

where CN_v^t is the set of t-type context nodes of v such first/second order neighbors [29] in the graph or local neighbors in short random walks [20]. The conditional probability $p(v_c|v;\Theta)$ is defined as the heterogeneous softmax function:

$$p(v_c|v;\Theta) = \frac{exp\left\{\mathcal{E}_{v_c} \cdot \mathcal{E}_v\right\}}{\sum_{v_k \in V_t} exp\left\{\mathcal{E}_{v_k} \cdot \mathcal{E}_v\right\}},\ \mathcal{E}_v = \mathcal{F}_\Theta(v) \tag{8}$$

where V_t is the set of t-type nodes in the graph, $\mathcal{E}_v$ is the output node embedding formulated by the proposed graph neural network Eq. (6) with all neural network parameters Θ. We leverage the negative sampling technique (NS) [19] to optimize the objective o_1 in Eq. (7). Specifically, by applying NS to the construction of softmax function in Eq. (8), we can approximate the logarithm of $p(v_c|v;\Theta)$ as:

$$\log \sigma(\mathcal{E}_{v_c} \cdot \mathcal{E}_v) + \sum_{m=1}^{M} \mathbb{E}_{v_{c'} \sim P_t(v_{c'})} \log \sigma(-\mathcal{E}_{v_{c'}} \cdot \mathcal{E}_v) \tag{9}$$

where M is the negative sample size and $P_t(v_{c'})$ is the pre-defined noise distribution $w.r.t.$ the t-type nodes. In this model, we set $M = 1$ as it makes little impact when $M > 1$. Thus Eq. (9) degenerates to the cross entropy loss:

$$\log \sigma(\mathcal{E}_{v_c} \cdot \mathcal{E}_v) + \log \sigma(-\mathcal{E}_{v_{c'}} \cdot \mathcal{E}_v) \tag{10}$$

In other words, for each context node v_c of v, we sample a negative node $v_{c'}$ according to $P_t(v_{c'})$. Therefore, we can reformulate the objective o_1 in Eq. (7) as follows:

$$o_2 = \sum_{\langle v, v_c, v_{c'} \rangle \in T_{walk}} \log \sigma(\mathcal{E}_{v_c} \cdot \mathcal{E}_v) + \log \sigma(-\mathcal{E}_{v_{c'}} \cdot \mathcal{E}_v) \tag{11}$$

where T_{walk} denotes the set of triplets $\langle v, v_c, v_{c'} \rangle$ collected by walk sampling on the graph. Similar to DeepWalk [20], we employ the random walk to generate T_{walk}. Specifically, first, we uniformly generate a set of random walks S in the heterogeneous graph. Then, for each node v in a walk $S_i \in S$, we collect context node v_c which satisfies: $dist(v, v_c) \leq \tau$, $i.e.$, v_c is within distance τ to v in S_i. Finally, for each v_c, we sample a negative node $v_{c'}$ with the same node type of v_c according to $P_t(v_{c'}) \propto dg_{v_{c'}}^{3/4}$, where $dg_{v_{c'}}$ is the frequency of $v_{c'}$ in S. To optimize the model parameters of HetGNN, at each iteration, we first sample a mini-batch of triplets in T_{walk} and accumulate the objective according to Eq. (11). Then, we update the model parameters via the Adam optimizer [11]. We repeat the training iterations until the change between two consecutive iterations is sufficiently small (see Section A.1 in supplement for a pseudocode of this training procedure). With the learned model parameters, we can infer node representations $\mathcal{E}$ for various graph mining tasks, as we will show in Section 4.

4 EXPERIMENTS

In this section, we conduct extensive experiments with the aim of answering the following research questions:

- (**RQ1**) How does HetGNN perform $vs.$ state-of-the-art baselines for various graph mining tasks, such as link prediction (**RQ1-1**), personalized recommendation (**RQ1-2**), and node classification & clustering (**RQ1-3**)?
- (**RQ2**) How does HetGNN perform $vs.$ state-of-the-art baselines for inductive graph mining tasks, such as inductive node classification & clustering?
- (**RQ3**) How do different components, $e.g.$, node heterogeneous contents encoder or heterogeneous neighbors aggregator, affect the model performance?
- (**RQ4**) How do various hyper-parameters, $e.g.$, embedding dimension or the size of sampled heterogeneous neighbors set, impact the model performance?

4.1 Experiment Design

4.1.1 Datasets.
We use four datasets of two kinds of HetG: academic graph and review graph. For the academic graph, we extract two datasets, $i.e.$, A-I contains papers between year 1996 and 2005 and A-II contains papers between year 2006 and 2015), from the public AMiner [30] data[1]. For the review graph, we extract two datasets, $i.e.$, R-I (Movies category) and R-II (CDs category), from the public Amazon [8] data[2]. The main statistics of four datasets are summarized in

[1] https://aminer.org/data
[2] http://jmcauley.ucsd.edu/data/amazon/index.html

Table 2: Datasets used in this work.

Data	Node	Edge
Academic I (A-I)	# author: 160,713 # paper: 111,409 # venue: 150	# author-paper: 295,103 # paper-paper: 138,464 # paper-venue: 111,409
Academic II (A-II)	# author: 28,646 # paper: 21,044 # venue: 18	# author-paper: 69,311 # paper-paper: 46,931 # paper-venue: 21,044
Movies Review (R-I)	# user: 18,340 # item: 56,361	# user-item: 629,125
CDs Review (R-II)	# user: 16,844 # item: 106,892	# user-item: 555,050

Table 2 (see Section A.2 in supplement for detail of these datasets). Note that HetGNN is flexible to be applied to other HetG.

4.1.2 Baselines.

We use five baselines including heterogeneous graph embedding model **metapath2vec** [4] (represented as **MP2V**), attributed graph models **ASNE** [15] and **SHNE** [34], as well as graph neural network models **GraphSAGE** [7] (represented as **GSAGE**) and **GAT** [31] (see Section A.3 in supplement for detailed settings of these baseline methods).

4.1.3 Reproducibility.

For the proposed model, the embedding dimension is set as 128. The size of sampled neighbor set (in Section 3.1) equals 23 (10, 10, 3 for author, paper, venue neighbor groups, respectively) in academic data. This value equals 20 (10, 10 for user, item neighbor groups, respectively) in review data. We use Par2Vec [19] and CNN [17] to pre-train text and image features, respectively. Besides, the DeepWalk [20] is employed to pre-train node embeddings. The nodes in academic data are associated with text (paper abstract) features and pre-trained node embeddings, while the nodes in review data include text (item description), image (item picture) features, and pre-trained node embeddings. Section A.4 of supplement contains more detailed settings. We employ Pytorch[3] to implement HetGNN and conduct experiments on GPU. Code is available at: https://github.com/chuxuzhang/KDD2019_HetGNN.

4.2 Applications

*4.2.1 Link Prediction (**RQ1-1**).*

Which links will happen in the future? To answer **RQ1-1**, we design experiments to evaluate HetGNN on several link prediction tasks.

Setting. Unlike previous work [6] that randomly samples a portion of links for training and uses the remaining for evaluation, we consider a more practical setting that splits training and test data sequentially. Specifically, first, the graph of training data is utilized to learn node embeddings and the corresponding links are used to train a binary logistic classifier. Then, test relations with equal number of random negative (non-connected) links are used to evaluate the trained classifier. In addition, only new links among nodes in training data are considered and duplicated links are removed from evaluation. The link embedding is formed by element-wise multiplication of embeddings of the two edge nodes. We use **AUC** and **F1** scores as evaluation metrics. In academic data, we consider two types of links: (type-1) collaboration between two authors and (type-2) citation between author and paper. The data

[3]https://pytorch.org/

Table 3: Link prediction results. Split notation in data denotes train/test data split years or ratios.

$Data_{split}$	Metric	MP2V [4]	ASNE [15]	SHNE [34]	GSAGE [7]	GAT [31]	HetGNN
$A\text{-}I_{2003}$ (type-1)	AUC	0.636	0.683	0.696	0.694	0.701	**0.714**
	F1	0.435	0.584	0.597	0.586	0.606	**0.620**
$A\text{-}I_{2003}$ (type-2)	AUC	0.790	0.794	0.781	0.790	0.821	**0.837**
	F1	0.743	0.774	0.755	0.746	0.792	**0.815**
$A\text{-}I_{2002}$ (type-1)	AUC	0.626	0.667	0.688	0.681	0.691	**0.710**
	F1	0.412	0.554	0.590	0.567	0.589	**0.615**
$A\text{-}I_{2002}$ (type-2)	AUC	0.808	0.782	0.795	0.806	0.837	**0.851**
	F1	0.770	0.753	0.761	0.772	0.816	**0.828**
$A\text{-}II_{2013}$ (type-1)	AUC	0.596	0.689	0.683	0.695	0.678	**0.717**
	F1	0.348	0.643	0.639	0.615	0.613	**0.669**
$A\text{-}II_{2013}$ (type-2)	AUC	0.712	0.721	0.695	0.714	0.732	**0.767**
	F1	0.647	0.713	0.674	0.664	0.705	**0.754**
$A\text{-}II_{2012}$ (type-1)	AUC	0.586	0.671	0.672	0.676	0.655	**0.701**
	F1	0.318	0.615	0.612	0.573	0.560	**0.642**
$A\text{-}II_{2012}$ (type-2)	AUC	0.724	0.726	0.706	0.739	0.750	**0.775**
	F1	0.664	0.737	0.692	0.706	0.715	**0.757**
$R\text{-}I_{5:5}$	AUC	0.634	0.623	0.651	0.661	0.683	**0.749**
	F1	0.445	0.551	0.586	0.542	0.665	**0.735**
$R\text{-}I_{7:3}$	AUC	0.701	0.656	0.695	0.716	0.706	**0.787**
	F1	0.595	0.613	0.660	0.688	0.702	**0.776**
$R\text{-}II_{5:5}$	AUC	0.678	0.655	0.685	0.677	0.712	**0.736**
	F1	0.541	0.582	0.593	0.565	0.659	**0.701**
$R\text{-}II_{7:3}$	AUC	0.737	0.695	0.728	0.721	0.742	**0.772**
	F1	0.660	0.648	0.685	0.653	0.713	**0.749**

before T_s (split year) is training data, otherwise test data. T_s of A-I data is set to 2003 and 2002. The value for A-II data is set to 2013 and 2012. In the review data, we consider user-item review links and divide training/test data sequentially. The train/test ratio (in terms of review number) is set to $7:3$ and $5:5$ for both R-I and R-II data.

Result. The performances of all models are reported in Table 3, where the best results are highlighted in bold. According to this table: (a) the best baselines in most cases are attributed graph embedding methods or graph neural network models, showing that incorporating node attributes or employing deep neural network generates desirable node embeddings for link prediction; (b) HetGNN outperforms all baselines in all cases especially in review data. The relative improvements (%) over the best baselines range from 1.5% to 5.6% and 3.4% to 10.5% for academic data and review data, respectively. It demonstrates that the proposed heterogeneous graph neural network framework is effective and obtains better node embeddings (than baselines) for link prediction.

*4.2.2 Recommendation (**RQ1-2**).*

Which nodes should be recommended to the target node? To answer **RQ1-2**, we design experiment to evaluate HetGNN on personalized node recommendation task.

Setting. The concept of node recommendation is similar to link prediction besides the experimental settings and evaluation metrics. To distinguish with the previous link prediction task, we evaluate venue recommendation (author-venue link) performance in the academic data. Specifically, the graph in training data is utilized to learn node embeddings. The ground truth of recommendation is based on author's appearance (having papers) in venue of test

Table 4: Recommendation results. Split notation in data denotes train/test data split years.

$Data_{split}$	Metric	MP2V [4]	ASNE [15]	SHNE [34]	GSAGE [7]	GAT [31]	HetGNN
$A\text{-}I_{2003}$	Rec	0.158	0.201	0.298	0.263	0.275	**0.319**
	Pre	0.044	0.060	0.081	0.077	0.079	**0.094**
	F1	0.069	0.092	0.127	0.120	0.123	**0.145**
$A\text{-}I_{2002}$	Rec	0.144	0.152	0.279	0.231	0.274	**0.293**
	Pre	0.046	0.050	0.086	0.073	0.087	**0.093**
	F1	0.070	0.075	0.134	0.112	0.132	**0.141**
$A\text{-}II_{2013}$	Rec	0.516	0.419	0.608	0.540	0.568	**0.625**
	Pre	0.207	0.174	0.241	0.219	0.230	**0.252**
	F1	0.295	0.333	0.345	0.312	0.327	**0.359**
$A\text{-}II_{2012}$	Rec	0.468	0.382	0.552	0.512	0.518	**0.606**
	Pre	0.204	0.171	0.233	0.224	0.227	**0.264**
	F1	0.284	0.236	0.327	0.312	0.316	**0.368**

Table 5: Multi-label classification (MC) and node clustering (NC) results. Percentage denotes training data ratio.

Task	Metric	MP2V [4]	ASNE [15]	SHNE [34]	GSAGE [7]	GAT [31]	HetGNN
MC (10%)	Macro-F1	0.972	0.965	0.939	**0.978**	0.962	**0.978**
	Micro-F1	0.973	0.967	0.940	0.978	0.963	**0.979**
MC (30%)	Macro-F1	0.975	0.969	0.939	0.979	0.965	**0.981**
	Micro-F1	0.975	0.970	0.941	0.980	0.965	**0.982**
NC	NMI	0.894	0.854	0.776	**0.914**	0.845	0.901
	ARI	0.933	0.898	0.813	**0.945**	0.882	0.932

data. The preference score is defined as the inner-product between embeddings of two nodes. We use Recall (**Rec**), Precision (**Pre**), and **F1** scores in top-k recommendation list as the evaluation metric. In addition, duplicated author-venue pairs are removed from evaluation. The reported score is the average value over all evaluated authors. The same as link prediction task, the train/test split year T_s for A-I data is set to 2003 and 2002. The value for A-II data is set to 2013 and 2012. Besides, k is set to 5 and 3 for two data respectively.

Result. The results of different models are reported in Table 4. The best results are highlighted in bold. According to this table, the best baselines are attributed graph embedding methods or graph neural network models in most cases. In addition, HetGNN performs best in all cases. The relative improvements (%) over the best baseline range from 2.8% to 16.0%, showing that HetGNN is effective and can learn better node embeddings (than baselines) for node recommendation.

*4.2.3 Classification and Clustering (**RQ1-3**).*

Which class/cluster does this node belong to? To answer **RQ1-3**, we design experiments to evaluate HetGNN for multi-labels classification and node clustering tasks.

Setting. Similar to metapath2vec [4], we match authors in A-II dataset with four selected research domains, *i.e.*, Data Mining (DM), Computer Vision (CV), Natural Language Processing (NLP) and Database (DB). Specifically, we choose three top venues[4] for each area. Each author is labeled with the area with the majority of his/her publications (authors without paper in these venues are excluded in evaluation). The node embeddings are learned from the full dataset. For the multi-labels classification task, the learned node

[4]DM: KDD, WSDM, ICDM. CV: CVPR, ICCV, ECCV. NLP: ACL, EMNLP, NAACL. DB: SIGMOD, VLDB, ICDE

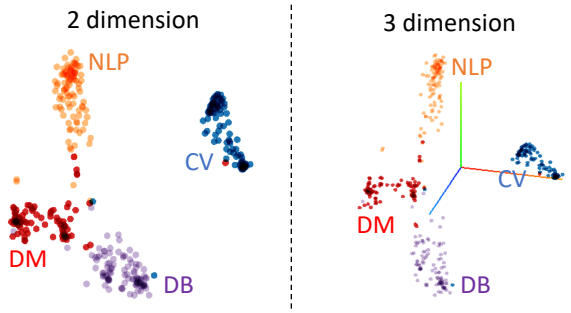

Figure 3: Author embeddings visualization of four selected domains in academic data.

Table 6: Inductive multi-labels classification (IMC) and node clustering (INC) results. Percentage is training data ratio.

Task	Metric	GSAGE [7]	GAT [31]	HetGNN
IMC (10%)	Macro-F1	0.938	0.954	**0.962**
	Micro-F1	0.945	0.958	**0.965**
IMC (30%)	Macro-F1	0.949	0.956	**0.964**
	Micro-F1	0.955	0.960	**0.968**
INC	NMI	0.714	0.765	**0.840**
	ARI	0.764	0.803	**0.894**

embeddings are used as the input to a logistic regression classifier. Besides, the size (ratio) of training data is set to 10% and 30%, and the remaining nodes are used for test. We use both **Micro-F1** and **Macro-F1** as evaluation metrics. For the node clustering task, the learned node embeddings are used as the input to a clustering model. Here we employ the k-means algorithm to cluster the data and evaluate the clustering performance in terms of normalized mutual information (**NMI**) and adjusted rand index (**ARI**).

Result. Table 5 reports results of all methods, where the best results are highlighted in bold. It is can be seen that: (1) most of models have good performance in multi-labels classification and obtain large Macro-F1 and Micro-F1 scores (over 0.95). It is reasonable since authors of four selected domains are quite different from each other; (2) Despite (1), HetGNN achieves the best performance or is comparable to the best method for multi-label classification and node clustering tasks, showing that HetGNN can learn effective node embeddings for these tasks.

Furthermore, we employ TensorFlow embedding projector to visualize author embeddings of four domains, as shown by Figure 3. For each area, we randomly sample 100 authors. It is easy to see that embeddings of authors in the same class cluster closely and can be well distinguished from others in both 2D and 3D visualizations, demonstrating the effectiveness of learned node embeddings.

*4.2.4 Inductive Classification and Clustering (**RQ2**).*

Which class/cluster does new node belong to? To answer **RQ2**, we design experiment to evaluate HetGNN for inductive multi-labels classification and inductive node clustering tasks.

Setting. The setting of this task is similar to the previous node classification and clustering tasks except that we use the new node embeddings as the model input. Specifically, first, we use the training data (A-II dataset, train/test split year = 2013) to train the model. Then, we employ the learned model to infer the embeddings of

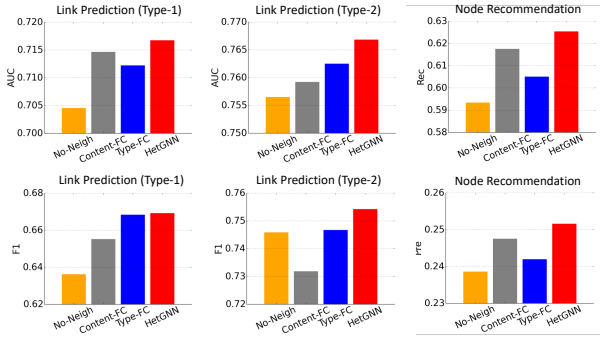

Figure 4: Performances of variant proposed models.

all new nodes in test data. Finally, we use the inferred new node embeddings as the input to classification and clustering models.

Result. Table 6 reports performances of graph neural network models, where the best results are highlighted in bold. According to this table: (1) all methods have good performances in inductive multi-labels classification as the reason described in the previous task. However, HetGNN still achieves the best performance; (2) The result of HetGNN is better than the others for inductive node clustering. The average relative improvements (%) over GSAGE and GAT are 17.3% and 10.6%, respectively. It shows that the learned HetGNN model is effective for inferring new node embeddings.

4.3 Analysis

*4.3.1 Ablation Study (**RQ3**).*

HetGNN is a joint learning framework of node heterogeneous contents encoding and heterogeneous neighbors aggregation. How content encoder impact the model performance? Whether neighbors aggregation is effective for improving the model capability? To answer these questions and **RQ3**, we conduct ablation studies to evaluate performances of several model variants which include: (a) **No-Neigh** that uses heterogeneous contents encoding to represent each node embedding (without neighbors information); (b) **Content-FC** that employs a fully connected neural network (FC) to encode node heterogeneous contents; (c) **Type-FC** that utilizes a FC to combine embeddings of different neighbor types (see Section A.5 in supplement for detail of model variants). The results of link prediction and node recommendation on A-II dataset (train/test split year = 2013) are reported in Figure 4. From this figure:

- HetGNN has better performance than No-Neigh in most cases, demonstrating that aggregating neighbors information is effective for generating better node embeddings.
- HetGNN outperforms Content-FC, indicating that the Bi-LSTM based content encoding is better than "shallow" encoding like FC for capturing "deep" content feature interactions.
- HetGNN achieves better results than Type-FC, showing that self-attention is better than FC for capturing node type impact.

*4.3.2 Hyper-parameters Sensitivity (**RQ4**).*

The hyper-parameters play important roles in HetGNN, as they determine how the node embeddings will be generated. We conduct experiments to analyze the impacts of two key parameters, *i.e.*, the embedding dimension d and the size of sampled neighbors set for each node (see Section A.6 in supplement for detailed setup). The link prediction and recommendation performances of HetGNN as

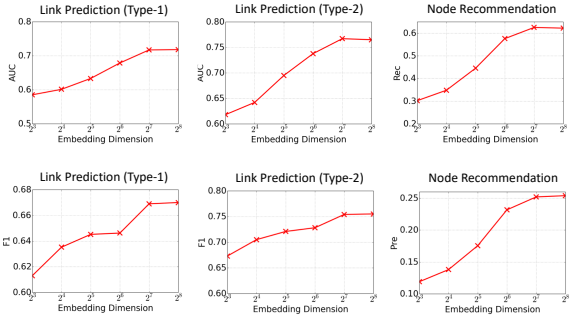

Figure 5: Impact of embedding dimension.

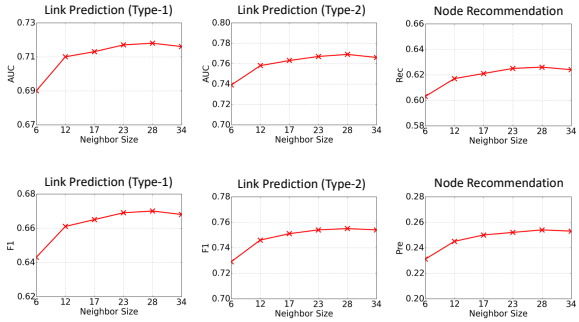

Figure 6: Impact of sampled neighbor size.

a function of embedding dimension and sampled neighbor size on A-II dataset (train/test split year = 2013) are shown in Figure 5 and Figure 6, respectively. According to these figures, we can find that:

- When d varies from 8 to 256, all evaluation metrics increase in general since better representations can be learned. However, the performance becomes stable or slightly worse when d further increases. This may due to over-fitting.
- When the neighbor size varies from 6 to 34, all evaluation metrics increase at first as suitable amount of neighborhood information are considered. But when the size of neighbors exceeds a certain value, performance decreases slowly which may due to uncorrelated ("noise") neighbors are involved. The best neighbor size is in the range of 20 to 30.

5 RELATED WORK

The related study includes: (1) heterogeneous graph mining; (2) graph representation learning; (3) graph neural networks.

Heterogeneous graph mining. In the past decade, many work have been devoted to mining heterogeneous graphs (HetG) for different applications such as relation inference [2, 25, 33, 35], personalized recommendation [10, 23], classification [36], *etc*. For example, Sun *et al*. [25] leveraged metapath based approach to extract topological features and predict citation relationship in academic graph. Chen *et al*. [2] designed a HetG based ranking model to identify authors of anonymous papers. Zhang *et al*. [36] proposed a deep convolutional classification model for collective classification in HetG.

Graph representation learning. Graph representation learning [3] has became one of the most popular data mining topics in the past few years. Graph structure based models [4, 6, 20, 29] were proposed to learn vectorized node embeddings that can be further

utilized in various graph mining tasks. For example, inspired by word2vec [19], Perozzi *et al*. [20] developed the innovative DeepWalk which introduces node-context concept in graph (analogy to word-context) and feeds a set of random walks over graph (analogy to "sentences") to SkipGram so as to obtain node embeddings. Later, to address graph structure heterogeneity, Dong *et al*. [4] introduced metapath guided walks and proposed metapath2vec for representation learning in HetG. Further, attributed graph embedding models [14, 15, 34] have been proposed to leverages both graph structure and node attributes for learning node embeddings. Besides those methods, many other approaches have been proposed [1, 18, 21, 28, 32], such as NetMF [21] that learns node embedding via matrix factorization and NetRA [32] that uses adversarially regularized autoencoders to learn node embeddings, and so on.

Graph neural networks. Recently, with the advent of deep learning, graph neural networks (GNNs) [5, 7, 12, 16, 24, 31] has gained a lot of attention. Unlike previous graph embedding models, the key idea behind GNNs is to aggregate feature information from node's local neighbors via neural networks. For example, GraphSAGE [7] uses neural networks, *e.g.*, LSTM, to aggregate neighbors' feature information. Besides, GAT [31] employs self-attention mechanism to measure impacts of different neighbors and combine their impacts to obtain node embeddings. Moreover, some task dependent approaches, *e.g.*, GEM [16] for malicious accounts detection, have been proposed to obtain better node embeddings for specific tasks.

6 CONCLUSION

In this paper, we introduced the problem of heterogeneous graph representation learning and proposed a heterogeneous graph neural network model, *i.e.*, HetGNN, to address this problem. HetGNN jointly considered node heterogeneous contents encoding, type-based neighbors aggregation, and heterogeneous types combination. In the training stage, a graph context loss and a mini-batch gradient descent procedure were employed to learn the model parameters. Extensive experiments on various graph mining tasks, *i.e.*, link prediction, recommendation, node classification & clustering and inductive node classification & clustering, demonstrated that HetGNN can outperform state-of-the-art methods.

ACKNOWLEDGMENTS

This work is supported by the CCDC Army Research Laboratory under Cooperative Agreement Number W911NF-09-2-0053 (Network Science CTA) and the National Science Foundation (NSF) grant IIS-1447795.

REFERENCES

[1] 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*. 119–128.
[2] Ting Chen and Yizhou Sun. 2017. Task-Guided and Path-Augmented Heterogeneous Network Embedding for Author Identification. In *WSDM*. 295–304.
[3] Peng Cui, Xiao Wang, Jian Pei, and Wenwu Zhu. 2018. A survey on network embedding. *TKDE* (2018).
[4] Yuxiao Dong, Nitesh V Chawla, and Ananthram Swami. 2017. metapath2vec: Scalable Representation Learning for Heterogeneous Networks. In *KDD*. 135–144.
[5] Hongyang Gao, Zhengyang Wang, and Shuiwang Ji. 2018. Large-scale learnable graph convolutional networks. In *KDD*. 1416–1424.
[6] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *KDD*. 855–864.
[7] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *NIPS*. 1024–1034.
[8] Ruining He and Julian McAuley. 2016. Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering. In *WWW*. 507–517.
[9] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.
[10] Binbin Hu, Chuan Shi, Wayne Xin Zhao, and Philip S Yu. 2018. Leveraging meta-path based context for top-n recommendation with a neural co-attention model. In *KDD*. 1531–1540.
[11] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).
[12] Thomas N Kipf and Max Welling. 2017. Semi-supervised classification with graph convolutional networks. In *ICLR*.
[13] Quoc Le and Tomas Mikolov. 2014. Distributed representations of sentences and documents. In *ICML*. 1188–1196.
[14] Jundong Li, Harsh Dani, Xia Hu, Jiliang Tang, Yi Chang, and Huan Liu. 2017. Attributed network embedding for learning in a dynamic environment. In *CIKM*. 387–396.
[15] Lizi Liao, Xiangnan He, Hanwang Zhang, and Tat-Seng Chua. 2018. Attributed social network embedding. *TKDE* 30, 12 (2018), 2257–2270.
[16] Ziqi Liu, Chaochao Chen, Xinxing Yang, Jun Zhou, Xiaolong Li, and Le Song. 2018. Heterogeneous Graph Neural Networks for Malicious Account Detection. In *CIKM*. 2077–2085.
[17] Jonathan Long, Evan Shelhamer, and Trevor Darrell. 2015. Fully convolutional networks for semantic segmentation. In *CVPR*. 3431–3440.
[18] Jianxin Ma, Peng Cui, Xiao Wang, and Wenwu Zhu. 2018. Hierarchical Taxonomy Aware Network Embedding. In *KDD*. 1920–1929.
[19] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *NIPS*. 3111–3119.
[20] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *KDD*. 701–710.
[21] Jiezhong Qiu, Yuxiao Dong, Hao Ma, Jian Li, Kuansan Wang, and Jie Tang. 2018. Network embedding as matrix factorization: Unifying deepwalk, line, pte, and node2vec. In *WSDM*. 459–467.
[22] Meng Qu, Jian Tang, and Jiawei Han. 2018. Curriculum Learning for Heterogeneous Star Network Embedding via Deep Reinforcement Learning. In *WSDM*. 468–476.
[23] Xiang Ren, Jialu Liu, Xiao Yu, Urvashi Khandelwal, Quanquan Gu, Lidan Wang, and Jiawei Han. 2014. Cluscite: Effective citation recommendation by information network-based clustering. In *KDD*. 821–830.
[24] 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*. 593–607.
[25] Yizhou Sun, Jiawei Han, Charu C Aggarwal, and Nitesh V Chawla. 2012. When will it happen?: relationship prediction in heterogeneous information networks. In *WSDM*. 663–672.
[26] Yizhou Sun, Jiawei Han, Xifeng Yan, Philip S Yu, and Tianyi Wu. 2011. Pathsim: Meta path-based top-k similarity search in heterogeneous information networks. *VLDB* 4, 11 (2011), 992–1003.
[27] Yizhou Sun, Brandon Norick, Jaiwei Han, Xifeng Yan, Philip Yu, and Xiao Yu. 2012. PathSelClus: Integrating Meta-Path Selection with User-Guided Object Clustering in Heterogeneous Information Networks. In *KDD*. 1348–1356.
[28] Jian Tang, Meng Qu, and Qiaozhu Mei. 2015. Pte: Predictive text embedding through large-scale heterogeneous text networks. In *KDD*. 1165–1174.
[29] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. Line: Large-scale information network embedding. In *WWW*. 1067–1077.
[30] Jie Tang, Jing Zhang, Limin Yao, Juanzi Li, Li Zhang, and Zhong Su. 2008. Arnetminer: extraction and mining of academic social networks. In *KDD*. 990–998.
[31] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2018. Graph attention networks. In *ICLR*.
[32] Wenchao Yu, Cheng Zheng, Wei Cheng, Charu C Aggarwal, Dongjin Song, Bo Zong, Haifeng Chen, and Wei Wang. 2018. Learning Deep Network Representations with Adversarially Regularized Autoencoders. In *KDD*. 2663–2671.
[33] Chuxu Zhang, Chao Huang, Lu Yu, Xiangliang Zhang, and Nitesh V Chawla. 2018. Camel: Content-Aware and Meta-path Augmented Metric Learning for Author Identification. In *WWW*. 709–718.
[34] Chuxu Zhang, Ananthram Swami, and Nitesh V Chawla. 2019. SHNE: Representation Learning for Semantic-Associated Heterogeneous Networks. In *WSDM*. 690–698.
[35] Chuxu Zhang, Lu Yu, Xiangliang Zhang, and Nitesh V Chawla. 2018. Task-Guided and Semantic-Aware Ranking for Academic Author-Paper Correlation Inference.. In *IJCAI*. 3641–3647.
[36] Yizhou Zhang, Yun Xiong, Xiangnan Kong, Shanshan Li, Jinhong Mi, and Yangyong Zhu. 2018. Deep Collective Classification in Heterogeneous Information Networks. In *WWW*. 399–408.

A SUPPLEMENT

A.1 Pseudocode of HetGNN Training Procedure

The pseudocode of HetGNN training procedure is described in Algorithm 1. The content features are pre-trained by different techniques and T_{walk} is collected by random walk sampling in graph (Section 3.4). After training, the optimized model parameters Θ can be utilized to infer node embeddings $\mathcal{E}$ which can be further used in various graph mining tasks.

Algorithm 1: Training Procedure of HetGNN

input : pre-trained content features of $v \in V$, triplets set T_{walk}
output : optimized model parameters Θ (for inferring node embeddings $\mathcal{E}$)

```
while not done do
    sample a batch of (v, v_c, v_c') in T_walk
    formulate embeddings of v, v_c, and v_c' by Eq. (6)
    accumulate the objective by Eq. (11)
    update the parameters Θ by Adam
end
return optimized Θ
```

A.2 Dataset Description

We use four datasets of two types of heterogeneous graphs: academic graph and review graph. For the academic graph, we extract two datasets from the public AMiner [30] data[5]. The first one (represented as A-I) contains publications information of the major computer science venues from year 1996 to 2005. In addition, considering most of researchers pay attention to papers published in top venues, we extract the second one (represented as A-II) which includes publications in a number of selected top venues[6] related to artificial intelligence and data science from year 2006 to 2015. Each paper has various bibliographic content information: title, abstract, authors, references, year, venue. For the review graph, we extract two datasets from the public Amazon [8] data (Movies category and CDs category)[7]. The dataset contains user review and item metadata from Amazon spanning from 05/1996 to 07/2014. Each item has various content information: title, description, genre, price, and picture.

A.3 Baseline Description

We use five baseline methods which include heterogeneous graph and attributed graph embedding models, as well as graph neural network models.

- **metapath2vec** [4]: It is a heterogeneous graph embedding model which leverages metapath guided walks and Skip-gram model to learn node embeddings.
- **ASNE** [15]: It is an attributed graph embedding method that uses both node "latent" features and attributed features to learn node embeddings.
- **SHNE** [34]: It jointly optimizes graph structure closeness and text semantic correlation to learn node embedding in text-associated heterogeneous graphs.
- **GraphSAGE** [7]: It is a graph neural network model that aggregates feature information of neighbors by different neural networks, such as LSTM.
- **GAT** [31]: It is a graph attention network model that aggregates neighbors' feature information by self-attention neural network.

A.4 Reproducibility Settings

The detailed settings for reproducing experiments in this work include:

- **Hyper-parameters.** The embedding dimension of HetGNN is set to 128. In Section 3.1, the return probability of RWR is set to 0.5 and the length of RWR for node $v \in V$ ($|\text{RWR}(v)|$) equals 100. The size of sampled neighbors set for each node equals 23 and 20 in academic data and review data, respectively. To be more specific, sizes of different neighbor groups (types) are 10 (author), 10 (paper), 3 (venue) in academic data, and 10 (user), 10 (item) in review data. In addition, we use random walk sampling to get the triplets set T_{walk} of the graph context loss in Section 3.4. The number of random walks rooted at each node equals 10, the walk length is set to 30, the window distance τ equals 5 for both data.
- **Content features.** In academic data, we use Par2Vec [19] to pre-train paper title and abstract contents. Besides, the DeepWalk [20] is employed to pre-train embeddings of author, paper, venue nodes based on the academic heterogeneous graph. The author node is associated with pre-trained author embedding, average abstract and title embeddings of some sampled papers that are written by the author. Thus the Bi-LSTM length of author content encoder equals 3. The paper node carries pre-trained paper embedding, title embedding, abstract embedding, average of its authors' pre-trained embeddings, and pre-trained embeddings of its venue. Therefore, the Bi-LSTM length of paper content encoder is 5. The venue node contains pre-trained venue embedding, average abstract and title embeddings of some sampled papers that are included in the venue. In other words, the Bi-LSTM length of venue content encoder equals 3. In review data, we use Par2Vec to pre-train item title and description content. The CNN [17] is utilized to pre-train item image (picture). Besides, DeepWalk is employed to get pre-trained embeddings of user and item nodes based on user-item review graph. The user node is associated with pre-trained user embedding, average description and image embeddings of items that are reviewed by the user. Thus the Bi-LSTM length of user content encoder is 3. Besides, the item node includes pre-trained item embedding, description embedding, and image embedding. In other words, the Bi-LSTM length of item content encoder equals 3.
- **Baseline settings.** For fair comparison, the embedding dimension d of all baselines are set to 128 (same as HetGNN). For MP2V, we employ three metapaths, *i.e.*, APA (author-paper-author), APVPA (author-paper-venue-paper-author) and APPA (author-paper-paper-author), and one metapath, *i.e.*, UIU (user-item-user), in academic and review data, respectively. Besides, the number of walks rooted at each node equals 10 and the walk length is set to 30 (same as the training procedure of HetGNN). For ASNE,

[5] https://aminer.org/data
[6] ICML, AAAI, IJCAI, CVPR, ICCV, ECCV, ACL, EMNLP, NAACL, KDD, WSDM, ICDM, SIGMOD, VLDB, ICDE, WWW, SIGIR, CIKM.
[7] http://jmcauley.ucsd.edu/data/amazon/index.html

besides "latent" feature, we use the same content features as HetGNN and concatenate them as general attribute features. For SHNE, we utilize paper abstract and item description (text sequence length = 100) as the input for deep semantic encoding (*i.e.*, LSTM) in two data, respectively. Besides, the walk sampling setting is the same as MP2V. For GraphSAGE and GAT, we use the same input features (concatenated as a general feature) and the sampled neighbors set for each node as HetGNN.

- **Software & Hardware.** We employ Pytorch[8] to implement HetGNN and further conduct it on a server with GPU machines. Code is available at: https://github.com/chuxuzhang/KDD2019_HetGNN.

A.5 Model Variants Description

In Section 4.3.1, we propose three model variants to conduct ablation study experiments. These models are:

- **No-Neigh**. This variant does not consider neighbors influence and uses heterogeneous contents encoding $f_1(v)$ (Section 3.2) to represent embedding of node $v \in V$. That is, it removes heterogeneous neighbors aggregation module (Section 3.3) of HetGNN.
- **Content-FC**. This variant replaces heterogeneous content encoder (Bi-LSTM) of HetGNN with a fully connected neural network (FC). That is, the concatenated content feature is fed to a FC layer to get content embedding. The other modules are the same as HetGNN.
- **Type-FC**. This variant replaces types combination module (attention) of HetGNN with a FC. That is, the concatenated embedding of different neighbor groups (types) is fed to a FC layer to get aggregated embedding. The other modules are the same as HetGNN.

Besides, the training procedures of all model variants are the same as HetGNN.

A.6 Hyper-parameters Sensitivity Setup

In Section 4.3.2, we conduct experiments on A-II dataset (train/test split year = 2013) to study the impacts of two hyper-parameters: embedding dimension d and the sampled neighbors size for each node. We investigate a specific parameter by changing its value and fixing the others. Specifically, when fixing sampled neighbor size (*i.e.*, 23), we set different embedding dimension d (*i.e.*, 8, 16, 32, 64, 128, 256) of HetGNN and evaluate its performance for each dimension. Besides, when fixing embedding dimension (*i.e.*, 128), we set different sizes of sampled neighbors set (*i.e.*, 6, 12, 17, 23, 28, 34) for each node and evaluate HetGNN's performance for each size. The constitutions of different neighbors groups (types) for aforementioned sizes are: 6 = 2 (author) + 2 (paper) + 2 (venue), 12 = 5 (author) + 5 (paper) + 2 (venue), 17 = 7 (author) + 7 (paper) + 3 (venue), 23 = 10 (author) + 10 (paper) + 3 (venue), 28 = 12 (author) + 12 (paper) + 4 (venue), and 34 = 15 (author) + 15 (paper) + 4 (venue).

[8] https://pytorch.org/