HeteGraph: A Convolutional Framework for Graph Learning in Recommender Systems

Dai Hoang Tran*, Abdulwahab Aljubairy*, Munazza Zaib*,
Quan Z. Sheng*, Wei Emma Zhang†, Nguyen H. Tran‡, Khoa L.D. Nguyen§
*Department of Computing , Macquarie University, Sydney, Australia
†Faculty of Engineering, Computer and Mathematical Sciences, The University of Adelaide, Adelaide, Australia
‡School of Computer Science, The University of Sydney, Sydney, Australia
§ Data61, CSIRO, Sydney, Australia
{dai-hoang.tran, abdulwahab.aljubairy, munazza-zaib.ghori}@hdr.mq.edu.au, michael.sheng@mq.edu.au,
wei.e.zhang@adelaide.edu.au, nguyen.tran@sydney.edu.au, khoa.nguyen@data61.csiro.au

***Abstract*—With the explosive growth of online information, many recommendation methods have been proposed. This research direction is boosted with deep learning architectures, especially the recently proposed Graph Convolutional Networks (GCNs). GCNs have shown tremendous potential in graph embedding learning thanks to its inductive inference property. However, most of the existing GCN based methods focus on solving tasks in the homogeneous graph settings, and none of them considers heterogeneous graph settings. In this paper, we bridge the gap by developing a novel framework called *HeteGraph* based on the GCN principles. *HeteGraph* can handle heterogeneous graphs in the recommender systems. Specifically, we propose a sampling technique and a graph convolutional operation to learn high quality graph's node embeddings, which differs from the traditional GCN approaches where a full graph adjacency matrix is needed for the embedding learning. For evaluation, we design two models based on the *HeteGraph* framework to evaluate two important recommendation tasks, namely *item rating prediction* and *diversified item recommendations*. Extensive experiments show our *HeteGraph*'s encouraging performance on the first task and state-of-the-art performance on the second task.**

***Index Terms*—Recommender System, Heterogeneous Graph, Graph Convolutional Network**

I. Introduction

Online users nowadays exposed to the increasingly huge amount of information are facing the "paradox of choice", which leads to severe cognitive overload. Recommender systems have been developed and adopted as effective solutions, where users are recommended with the items tailored to their needs and preferences. Recommender system is an active research field spans across disciplines in data mining, machine learning, and human behaviour psychology. The early methods were mainly based on the principles of the collaborative filtering and the matrix factorization approaches [1].

However, in recent years, the rapid advancement in deep learning has contributed greatly to this field. Several new approaches using deep learning methods have been applied and seen fruitful results [2]–[4]. Particularly, the combination of deep learning and graph methodologies are commonly used to solve recommendation problems. The common workflow of this combined approach is illustrated in Fig. 1. The complete workflow contains multiple steps. Typically the raw data get converted to graph-structured data, then an embedding process will learn the graph's node embeddings, and use these embeddings in a deep neural network model to generate recommendations. A very crucial part of the workflow is the embedding method. A meaningful embedding can encode important properties of either the nodes, the edges, the local node's neighbourhoods or the entire graph depending on what specific applications we want to solve. For solving recommendation problems, we usually need to learn the node embeddings with the purpose of finding a group of similar nodes as recommendations. As such, a major component of solving recommendation tasks using deep learning model with graph is to have a *good embedding learning algorithm*. Especially, finding high quality embeddings that have low dimension from the graph entities is very important.

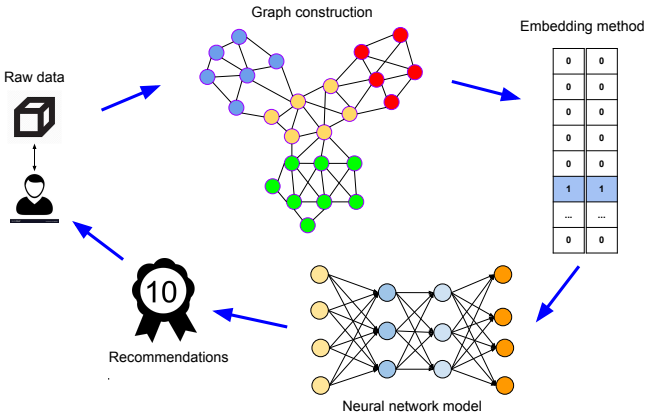

Fig. 1. Common workflow of deep learning recommender systems using graph-constructed data. Users, items and the interaction information are constructed as graph-structured data. Then the embedding method will derive the useful embeddings of those data which will be fed into deep neural network model. The final result are usually the top-k recommendations for a particular user.

Over the years, the research space of graph embedding has grown rapidly and several deep learning methods have been developed to learn graph-structure data embeddings effectively [5]. One popular approach that recently has spurred a lot of exciting developments is the deep learning architecture

978-1-7281-6926-2/20/$31.00 ©2020 IEEE

of Graph Convolutional Networks (GCNs) [6]. GCNs are neural network models that have been designed to work with graph-structured data using graph adjacency matrix. Based on GCNs, GraphSage [7] is another prominent method to learn graph-structured data via *graph convolutional operation* (GCO) and random walk sampling technique to reduce the memory footprint, as well as having the inductive learning capability. However, these proposed methods originally work with homogeneous graph data, while recommendation problems are mostly based on heterogeneous data types. The main challenge of working with heterogeneous data is the coalescing of multiple semantic data types into one uniform embedding. We give an example of a movie recommendation problem modelled as a heterogeneous graph. In this recommendation scenario, we have heterogeneous node types such as user nodes and movie nodes, each having their own attributes. The rating between a user and a movie is represented as the connected edge. Our objective of this recommendation problem is to predict which movies to be recommended to a user based on their previous rating interactions as well as their attributes.

Our work follows the new direction of using deep learning in solving recommendation problems. Particularly, we look at the problem of making recommendations as a link prediction task in graph, because we can construct recommendation data input as a bipartite heterogeneous graph. We present our work in handling heterogeneous graph-structured data of the recommendation problems based on the recent developments of GCN techniques. We aim to bridge the gap with current limitations of GCN techniques and heterogeneous data challenge by building a framework called HeteGraph. HeteGraph exploits the users' and items' attributes, their neighbourhood information, and the edge weight to learn useful embeddings, then feeds these embeddings into downstream recommendation tasks. With this novel architecture for learning heterogeneous node embeddings, we continue to tackle two important recommendation tasks.

The first recommendation task is about making a rating prediction between a user and an item, which is a fundamental task of a recommender system. Specifically, we want to show that using the GCO technique can help generate accurate rating predictions. The second recommendation task is about generating a diversified list of items to recommend to an active user. This second task has a different objective to the first one, which is to put more focus on the novelty of the generated recommendations. Novelty in the recommendations may give an active user a surprise as she discovers unanticipated items based on her previously interacted items, thus it might improve her satisfaction from the recommendations. In our evaluations, HeteGraph framework is able to make recommendations for both tasks with positive performance. Overall, our work on the HeteGraph framework allows us to contribute several aspects in solving recommendation problems based on graph convolutional principles. In a nutshell, the main contributions are as follows:

- *An adapted graph convolutional operation*. This operation is based on the GCN approaches to work with heterogeneous graph-structured data. This process includes the heterogeneous neighbourhood sampling and the adapted GCO.
- *A novel framework named HeteGraph*. Based on our adapted GCO, HeteGraph consists of four major phases and aims to solve challenging recommendation tasks.
- We conduct extensive experiments with two important recommendation tasks that have different objectives as a proof of concept to show how HeteGraph can help solve recommendation problems. The experiments based on two real-world datasets show the encouraging performance of HeteGraph on the first task and a state-of-the-art performance on the second task.

The rest of this paper is organized as follows. In Section II, we address the related works on recent researches of GCNs and its applications in recommender systems. We detail the HeteGraph framework architecture in Section III. We introduce our recommendation application models in Section IV. The evaluations are described in Section V, and we conclude our work in Section VI.

II. Related work

The works on embedding learning of graph-structured data have gained significant attention in recent years. The core mechanic of the graph embedding learning is to find a node embedding method to embed the node data into tensor form, then these tensors are applied into various downstream machine learning tasks. Graph Convolutional Network (GCNs) have been a rising trend recently for the task of learning node embedding. The terminology of *graph convolution* is originated by the seminal work of Bruna et al. [16], where the concept was developed based on special graph theory. Continuing this line of research, other researchers proposed improvements and extensions of this spectral convolution [6], [7], [17], and provided new state-of-the-art performance on benchmarking tasks such as node classification and link prediction. These successes have popularized these GCN algorithms, and researchers started to use them to solve other graph-structured data problems such as recommendation and drug-design tasks [21], [22]. Recommender systems have been traditionally relied on collaborative filtering approaches such as K-nearest neighbour or matrix factorization [1]. However, deep learning has changed the landscape of recommender algorithms. Researchers have used advanced models of deep learning to improve the recommender systems, such as autoencoder [3], recurrent neural network [2], and wide and deep model [24]. Henceforth, it is not straightforward to see new recommender's algorithms that leverage the convolution operation of the GCN's principles. However, these works either rely on spectral convolutional approach [23] or belong to a proprietary and specific service's task [25]. Therefore, our purpose in developing HeteGraph is to have a flexible and general framework that can handle common recommendation tasks, while leveraging the strong aspects of the GCO technique.

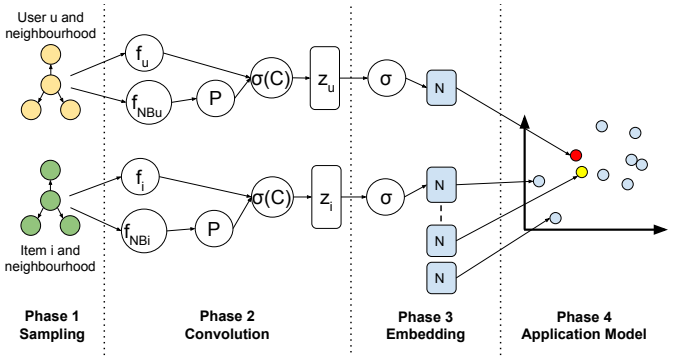

Fig. 2. The HeteGraph recommender framework for heterogeneous graph-structured data. f_u and f_{NB_u} are node u tensor and its neighbourhood tensor. P is the pooling layer. σ is the non-linear activation. C is the concatenation operation. Z_u is the convolved tensor. N is the node embedding. In Phase 1 we sample the node's neighbourhood. In Phase 2 we perform our GCO technique. In Phase 3 we learn the embeddings, and the embedding learning strategy depends on the application model of Phase 4.

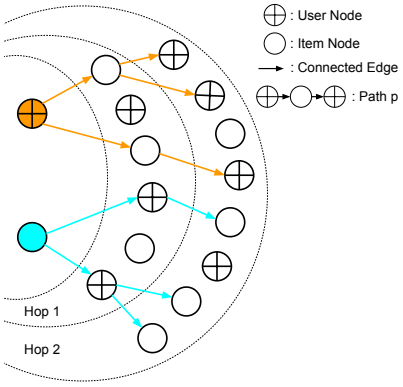

Fig. 3. HeteGrap neighbourhood sampling strategy. For a certain node of type χ, we perform a 2-hop random-walk to reach the next neighbour node of the same type χ. The walking path is biased by the connected edge's weight.

III. The HeteGraph Architecture

Fig. 2 shows the HeteGraph architecture with different phases. The overall strategy is that for every node in the graph, we sample its neighbourhood and learn the node embedding via our GCO technique. We can learn the embeddings either in supervised or unsupervised manner depending on the specific application model. These embeddings are then used as inputs to solve the targeted recommendation task. We will describe Phase 1 to 3 in this section, and Phase 4 "Application Model" will be described in Section IV.

A. Phase 1 - Neighbourhood Sampling

The learned embedding of each node in the HeteGraph model encodes both the node's attributes as well as its neighbourhood's attributes, thus we need to sample the node's neighbourhood first. Fig. 3 shows our sampling strategy using the random-walk technique. For a user node, we sample its neighbourhood by walking through an item node to reach the next user node via the connected edges. Similar strategy is applied for the item node. We use 2-hop walking distance in our model, since it gives the best performance in our algorithms based on the evaluations. The edge between a user-item node-pair during the walk represents their interaction. We leverage the edge weight to bias our walks when sampling. For example, in the case of movie recommendation problem, edge weight is the rating value. We define a relevant threshold δ. During the walk, edges with weight higher than δ are selected in random order. This helps HeteGraph to guide the random-walk process effectively. Therefore, for any node type, we can sample its neighbourhood with high relevance. This process is repeated several times for each node, forming a set of paths $NB_p = \{p_1, p_2, ..., p_i\}$ and each path p_i contains heterogeneous nodes $p_i = \{n_1, n_2, ..., n_k\}$. The cardinality $|NB_p|$ and $|p_i|$ are both predefined constants.

Algorithm 1 describes the implementation in pseudo code. First, we start the random-walk process by selecting the first target node n (line 7). Its neighbourhood is retrieved and sorted by connected edge weight in descending order (lines 9-10). To avoid getting fixed neighbourhood of node n for every sampling, we keep $k+c$ highest edge weight neighbourhood nodes, then select random k nodes from this "$k+c$ list" (lines 11-12). Finally, we select a random node rnb_i in the "k list" and add it to the i^{th} path of K sampling paths (line 13). The rnb_i is the next target node of the path-forming process (line 14) until we retrieve D nodes for this i^{th} path (line 5). This random-walk process is repeated until we sample K paths for the node n's neighbourhood NB_n (line 3).

Algorithm 1: HeteGraph Neighbourhood Sampling Algorithm (**NeighSamp**)

Input: node n of Graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$; Sampling size K; Walking distance D; Random elements retrival function $RAND$; Max path length constant k; Randomize constant c; Neighborhood retrieval function $NEIGHBOR$; Weight sort function $SORT$

Result: NB_n: The sampling neighbourhood of node n

1 **begin**
2 $NB_n \longleftarrow \emptyset$
3 **for** $i \leftarrow 1$ *to* K **do**
4 $target \leftarrow \emptyset$
5 **for** $j \leftarrow 1$ *to* D **do**
6 **if** j *equal* 1 **then**
7 $target \leftarrow n$
8 **end**
9 $\{nb_j\} = NEIGHBOR(target)$
10 $\{nb_j\}_{sort} \leftarrow SORT(\{nb_j\})$ in descending order
11 $\{nb_j\}_{filter} \leftarrow$ select first $k+c$ elements from $\{nb_j\}_{sort}$
12 $\{rand\_nb_j\} \leftarrow$ select k elements from $RAND(\{nb_j\}_{filter})$
13 $rnb_i \leftarrow RAND(\{rand\_nb_j\})$
14 $NB_n[i]$ adds rnb_i
15 $target \leftarrow rnb_i$
16 **end**
17 **end**
18 **return** NB_n
19 **end**

B. Phase 2 - Graph Convolutional Operation

After each node gets its sampled 2-hop neighbourhood, which is denoted by a set of bias random paths $NB_p = \{p_1, p_2, ..., p_i\}$, they become the inputs for the GCO. This GCO process is critical because it extracts the prominent characteristics of a node's neighbourhood, and puts them into the node embedding. Algorithm 2 describes our approach. The GCO involves two operational layers, the *neighbourhood attribute aggregation* (NAA) layer and the *attribute pooling* layer. The NAA layer takes each neighbourhood set of paths, transforms each path to an attributed tensor by combining all node's attributes in that path via concatenation operation. The final aggregated tensor of each neighbourhood is

$$f_{NB_n} = \begin{bmatrix} att_{p_1} \\ att_{p_2} \\ ... \\ att_{p_i} \end{bmatrix}, \quad (1)$$

where att_{p_i} is the combined attributes of all nodes n_k in path p_i of the neighbourhood NB_n (line 4). These neighbourhood tensors f_{NB} will be fed into the next *attribute pooling* layer. Similar to the pooling operation in a traditional convolutional neural network, our *attribute pooling* layer extracts the most prominent characteristics of the f_{NB}. Since it is very important that the *attribute pooling* layer's output is symmetric (permutation of its inputs will not change the output), we apply the symmetric pooling method called *mean pooling*:

$$f_{PO_n} = mean(\sigma(\mathbf{W}^{pool} \cdot f_{NB_n})), \quad (2)$$

$$mean(\begin{bmatrix} x_{11} & \dots & x_{1m} \\ \vdots & \ddots & \vdots \\ x_{n1} & \dots & x_{nm} \end{bmatrix}) = \begin{bmatrix} \frac{\sum_{i=1}^{n} x_{i1}}{n} & \dots & \frac{\sum_{i=1}^{n} x_{im}}{n} \end{bmatrix} \quad (3)$$

The *attribute pooling* layer comes with its own weight parameters $\mathbf{W}^{pool}$ and performs the mean operation of Equation (3) over the aggregated neighbourhood tensor f_{NB} to form the pooling neighbourhood tensor f_{PO_n} (line 5). Afterwards, the pooling neighbourhood tensor f_{PO_n} is concatenated with its original node attributes tensor (line 6). Finally, we apply non-linear activation function σ and normalization operation on the combined tensor, which we call the *convolved* tensor z, then use this z tensor in the embedding phase (lines 7-8).

C. Phase 3 - Embedding Learning Strategy

The embedding phase is the next step in the process of learning node embedding of the heterogeneous graph. We apply a particular loss method depending on the application model. For instance, if it is a supervised model then we can apply the *root mean square error* (RMSE) loss function to train this supervised model on every pair of user-item interactions.

$$L_{RMSE} = \sqrt{\frac{1}{n}\Sigma_{i=1}^{n}\Big(r_i - y_i\Big)^2}. \quad (4)$$

The RMSE value is also a standard measurement metric for the accuracy of recommendation problems based on explicit feedback. In Equation (4), r_i and y_i are the predicted rating and the actual rating respectively.

When we want to learn the node embeddings in an unsupervised manner, we apply an adapted version of hinge loss (HL) function with negative samplings to optimize the model parameters. For a user-item node-pair (u, i) that has high edge weight value, we want the dot product of their convolved tensor z_u and z_i to have a higher value than the dot product of the user-item node-pair (u, j), which has lower edge weight value (negative item).

$$L_{HL}(z_u z_i) = \mathbb{E}_{neg_u \sim P_{neg}(u)} \max\{0, z_u \cdot z_{neg_u} - z_u \cdot z_i + \Gamma\}. \quad (5)$$

Equation (5) shows our adapted HL function. $P_{neg}(u)$ is the probability distribution of the negative samplings for user u (irrelevant items to user u), Γ is the margin hyper-parameter and z_{neg_u} is the negative convolved tensor from the negative items sampling neg_u of user u. Our negative sampling technique is inspired by the work of T. Mikolov et al. [26]. We first select k random item nodes of the graph. Then, out of those k items, we remove the ones that have high edge weight value with user node n. The L_{HL} value is then optimized by using back-propagation with stochastic gradient descent method.

Algorithm 2: HeteGraph Graph Convolutional Operation Algorithm

Input: Graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$; node attributes $\{x_n, \forall n \in V\}$; Pooling weight matrix $\mathbf{W}^{pool}$; Convolution weight matrix $\mathbf{W}$; non-linearity function σ; Neighborhood sampling function `NeighSamp` (Algorithm 1); Neighbourhood attribute aggregation layer called $NEIGH\_AGG$; Attribute Pooling layer called $POOLING$; $CONCAT$ function to concatenate node's attributes;

Result: Graph Convolutional representation z_n for all $n \in \mathcal{V}$

1 **begin**
2 $Z \leftarrow \emptyset$
3 **for** $n \in \mathcal{V}$ **do**
4 $f_{NB_n} \leftarrow NEIGH\_AGG(\texttt{NeighSamp}(n))$
5 $f_{PO_n} \leftarrow POOLING(\sigma(\mathbf{W}^{pool} \cdot f_{NB_n}))$
6 $z_n \leftarrow \sigma(\mathbf{W} \cdot CONCAT(x_n, f_{PO_n}))$
7 $z_n \leftarrow z_n / \|z_n\|_2$
8 store z_n in set Z
9 **end**
10 **return** Z
11 **end**

IV. Recommendation Application Models

As mentioned in Section III-C, the embedding learning strategy of Phase 3 depends on the application model for a particular recommendation task. In our work, we aim to tackle two recommendation scenarios, which are 1) item rating prediction and 2) diversified item recommendations. We derive a supervised model for the first task, and an unsupervised model for the second task. We purposely choose these two different tasks with a contrasted way of model training to

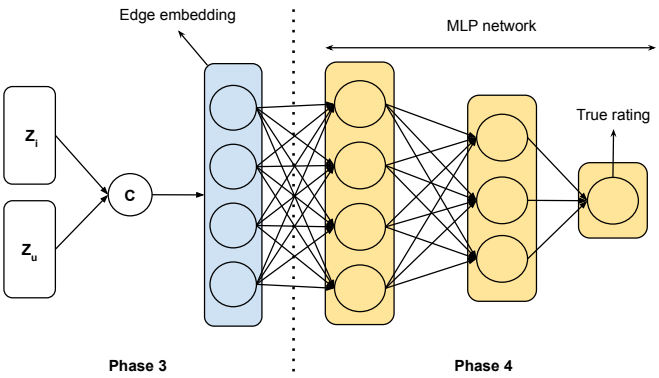

Fig. 4. HeteGraph Model 1 - Item rating prediction model. In this model, we use the edge embedding from the rating of a pair of user-item as input to the multilayer perceptron (MLP) neural network to train the rating prediction. The edge embedding is constructed via the concatenation operation C between the convolved tensor z_u of user u and the convolved tensor z_i of item i.

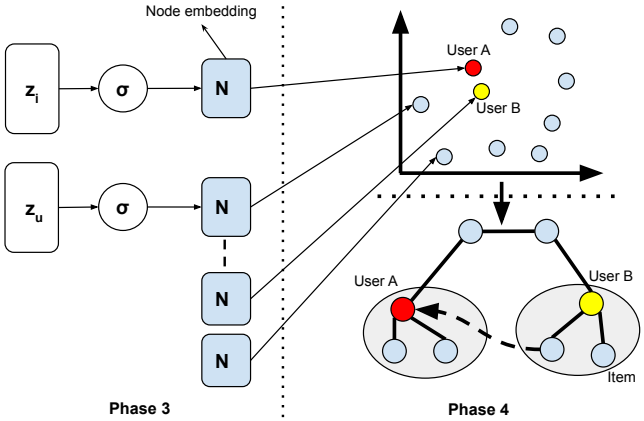

Fig. 5. HeteGraph Model 2 - Long distance search for diversified recommendations. In this model, user-A and user-B have node embeddings that are close to each other in the embedding space, thus they can have similar characteristics and preferences. However, in the original graph, they belong to different communities with a long distance between them. By recommending rated items from user-B to user-A, we can increase the diversity of the recommendations.

evaluate how effectively the GCO technique can be applied in solving recommendation problems. The detail of our application models is explained in this section.

A. Model 1 - Item Rating Predictions

Predicting item ratings for an active user is the most fundamental task of a recommender system, and based on these predicted ratings, a list of top-k relevant items is served to the active user. This is the typical feature of e-commerce/media consumption services such as book recommendations of Amazon [27] or movie recommendations of Netflix [28]. In these services, a user expresses her interest for an item via a rating score, usually in the numerical range such as from 1 to 5, whereas a low score provides the least favourable expression and a high score shows huge interest of the user to the item. From the graph perspective, we can consider this task as the link prediction or the link attribute inference problem between a *user node* and an *item node*. Given an unconnected user-item node-pair, we want to predict whether this node-pair should form an edge, based on the node-pair's attributes and its neighbourhood. In case of link attribute inference, we also want to predict the weigh value of this edge.

In our work, we build the model for the rating prediction problem by using the feed-forward neural network multilayer perceptron (MLP). As depicted in Fig. 4, for every known rating between a user-item, we represent its feature vector input as an edge embedding vector e_j. The edge embedding e_j is formed by concatenating the user convolved tensor z_{u_j} and item convolved tensor z_{i_j}. The input e_j is then fed into the MLP network, which can have multiple hidden layers and one output layer. The output layer of the MLP network contains the predicted rating. We use 2 hidden layers for this MLP network during our evaluation. Through Phase 1 to Phase 3 as explained in Section III, we learn the edge embedding as the model's input, and we use the true explicit rating of that user-item node-pair as the model's label output. The whole model is then optimized using RMSE loss to learn the shared weight parameters $\mathbf{W}$ between all training users' attributes and items' attributes. We denote this model as HeteGraph Supervised Model (*HeteGraph*$_{sup}$).

Noted that we use the RMSE loss directly from the final output to train our model, thus we cannot use the HL loss to learn the node embedding. Therefore, another variant of our model is proposed to solve this item rating prediction, denoted as HeteGraph Mix Model (*HeteGraph*$_{mix}$). With *HeteGraph*$_{mix}$, we first learn the node embeddings via HL loss in unsupervised manner as depicted in Phase 3 of Fig. 5, then we use those node embeddings to form the edge embeddings and go through the same process as described above to train this model. Since we use both unsupervised training to learn the node embeddings and supervised training for the rating prediction process in the this model variant, we use the word *mix* to differentiate our model variances.

B. Model 2 - Long Distance Search for Diversified Items

While the first model concerns about the HeteGraph performance on the typical task of a recommender system, which is rating prediction, we also would like to explore a new recommendation task in our second model. Our purpose is to see other practical aspects of graph convolution approach in a recommender system. Therefore, we derive a model to recommend items for their *diversity*. Diversity concept can be quite vague, thus in our application, we consider the diversity of items based on the items' attributes. All items are considered to have certain *common attributes* based on their local community. For instance, in the movie recommendation problem, a very common movie attribute is the *genre*. But if we look at the global scale of movies in different countries, another common attribute of movie is the *language*. Hence, if we want to recommend *relevant* movies, we recommend movies with similar common attributes such as both genre

and language. Any deviation of the common movie attributes are considered as *diversified* items. For instance, we now recommend two action movies but with different languages, that means those two items belong to different communities in the movie recommendation graph, and this is a diversified recommendation.

Our strategy to accomplish this task is to build a second model named *HeteGraph*$_{div}$ as depicted in Fig. 5. It is clear in Fig. 5 that first we use the unsupervised training in Phase 3 to learn the low dimensional node embedding of each user and item. The node embedding's position in its embedding space signifies its relationship with other nodes. Two embedding nodes that stay close to each other imply high similarity in their attributes as well as their local neighbourhood information, despite the fact that they may belong to different communities in the original graph. Phase 4 as illustrated in Fig. 5 describes the process. Given this intuition, we want to find users who are close in the embedding space but have a long distance in the original heterogeneous graph. Henceforth, their rated items can act as diversified recommendations to each other.

V. Evaluations

We evaluate the HeteGraph models with different metrics in recommendation accuracy and recommendation diversity. We compare them with popular matrix factorization and K-Nearest neighbourhood methods.

A. The Datasets and the Metrics

We perform evaluations on two datasets, the "MovieLens 100K" (ML-100K)[1] and "BookCrossing" (BX)[2]. The ML dataset contains 100,000 ratings of 1,682 movies from 943 users, each movie or user has its own respective attributes such as movie's genre, movie's title, user's age, and user's occupation. The BX dataset contains more than 1,000,000 ratings of books from its users, each also having their own attributes such as book's name, book's publisher, and user's location. One interesting aspect of the BX dataset is that it contains both implicit and explicit rating values. Value 0 means implicit rating, which can be interpreted as the user has an interaction with the book. Any other value above 0 is the explicit rating value. Table I summarizes the chosen datasets. Due to hardware limitation and the extreme sparseness of the BX dataset ratings (0.001%), we filter the BX dataset to include about 200,000 ratings of 5,102 users and 4,405 books, where each user has rated more than 30 books and each book is rated by more than 30 users. This helps increase the density of the BX dataset ratings to 0.98% as well as allow our hardware to process all of the data.

For the measurement of first task, we use four metrics including Mean Absolute Error (MAE) [30], Root Mean Square Error (RMSE), Precision-at-K (Pr@K) [29] and Recall-at-K (Re@K) [29]. The second task requires diversity metric, thus we adapt the Intra-list Simialry (ILS) from the work of [29]. Equation (6) is our adapted formula of the original ILS. For a recommendation list L of user u, the ILS score of user u is the summation of all similarity scores between item i_j and i_k in list L. Any similarity function sim can be used such as the cosine similarity or Jaccard similarity coefficient. In our evaluation, we use the cosine similarity function. The ILS score is then normalized by a factor of $2|L|$. The average ILS score of the whole test set is the average ILS_u scores of all users in the test set.

$$\text{ILS}_u = \frac{\sum_{i_j \in L} \sum_{i_k \in L} sim(i_j, i_k)}{2|L|} \tag{6}$$

[1] grouplens.org/datasets/movielens

[2] grouplens.org/datasets/book-crossing

TABLE I
Datasets for the evaluation

Dataset	ML-100K	BX-200K
User size	943	5,102
Item size	1,682	4,405
Rating size	100,000	219,289
Rating scale	1-5	0-10
Feature vector size	64	128
Rating density	6.3%	0.98%

B. Feature Preprocessing

As the GCO uses the node's attributes to learn the embeddings, we need to convert the raw attribute data of each node into a vectorized form. For text content such as item's title we use the "bag-of-words" technique for the conversion. Categorical values such as user's occupation of the ML dataset get converted into feature vector by using the one-hot-encoding technique. Other numerical values such as the user's age are normalized to centre around 0. Additionally, we pad extra 0s to either user's feature vector or item's feature vector to make them equal in size. For the user-item bipartite graph, we remove all isolated nodes to reduce the graph's size. Finally, we split the ratings (graph's edges) into training set and test set with a ratio of 80% and 20% respectively.

C. Model Evaluations

1) Accuracy of Item Rating Predictions (Model 1): The main evaluation metrics for item rating prediction is the accuracy. It is well known that accuracy metrics do not represent the overall satisfactory measurement of an active user when receiving recommendations [29]. However, it is still the most common evaluation metric used nowadays to assert certain confidence on the capability of a recommender system. We compare our model with other popular algorithms such as SVD and SVD++. The parameters for the Pr@k and Re@k evaluations are k, which is the size of recommendation list and the relevant threshold β, which is the value where both the predicted rating and the true rating values of a user-item pair must be higher, to be considered as relevant recommendation. For the ML-100K dataset, we set k as 20, and the threshold score β as 3.5 (rating scale from 1 to 5). For the BX dataset, we set k also as 20, and the threshold score β as 6 due to a different rating scale (from 0 to 10).

Table II shows our evaluation results. Surprisingly, both HeteGraph models achieve encouraging results. Admittedly,

TABLE II
ITEM RATING PREDICTION EVALUATION

Metrics	RMSE	MAE	Pr@20	Re@20
ML-100K dataset				
HeteGraph$_{sup}$	**0.976**	**0.761**	**0.737**	**0.679**
HeteGraph$_{mix}$	**0.958**	**0.733**	**0.741**	**0.664**
SVD [30]	0.943	0.743	0.763	0.659
SVD++ [30]	0.917	0.718	0.749	0.670
NMF [31]	0.963	0.763	0.728	0.637
SlopeOne [32]	0.950	0.745	0.733	0.654
kNN [33]	0.981	0.776	0.717	0.706
BX-200K dataset				
HeteGraph$_{sup}$	**3.702**	**2.802**	**0.843**	**0.388**
HeteGraph$_{mix}$	**3.634**	**2.776**	**0.857**	**0.403**
SVD	3.556	2.763	0.910	0.333
SVD++	3.796	2.793	0.808	0.392
NMF	3.885	2.785	0.743	0.438
SlopeOne	3.531	2.706	0.882	0.395
kNN	3.795	2.944	0.898	0.322

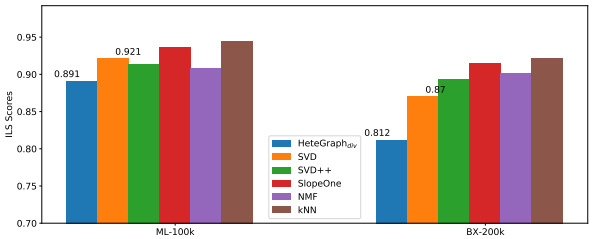

Fig. 6. Diversity Evaluation with ILS scores

both of them cannot outscore SVD or SVD++ in terms of RMSE and MAE metrics (lower is better), but we score higher than SVD++ in the BX dataset for the Pr@k and Re@k metrics (higher is better). Additionally, our models consistently perform better than kNN in the RMSE and MAE metrics in both datasets. For the comparison between our models variances HeteGraph$_{sup}$ and HeteGraph$_{mix}$, it is clear that HeteGraph$_{mix}$ scores better than HeteGraph$_{sup}$ in most of the evaluated metrics. This suggests that our unsupervised training model is able to extract useful node characteristics and can learn useful embeddings.

2) Diversity of Diversified Recommendations (Model 2): To verify the diversity of the recommendations, we use the intra-list similarity (ILS) score [29] for the diversification. ILS measures the diversity of a recommendation list, and lower score means more diversity. We modify the ILS score for each user by scaling down with a factor equal to that user's recommendation list size. Figure 6 shows the evaluation results of the HeteGraph$_{div}$ model. In terms of diversity, our model has the lowest ILS scores in both ML and BX datasets, which means we achieve the highest diversification in the recommendation list.

D. Hyper-parameters Analysis

During the evaluations, we conduct experiments with different learning rate α to find the best learning rate value for our models. In this section, we briefly discuss our observations. We make a grid-search attempt with different learning rate α, several optimizers and regularization techniques. We select five

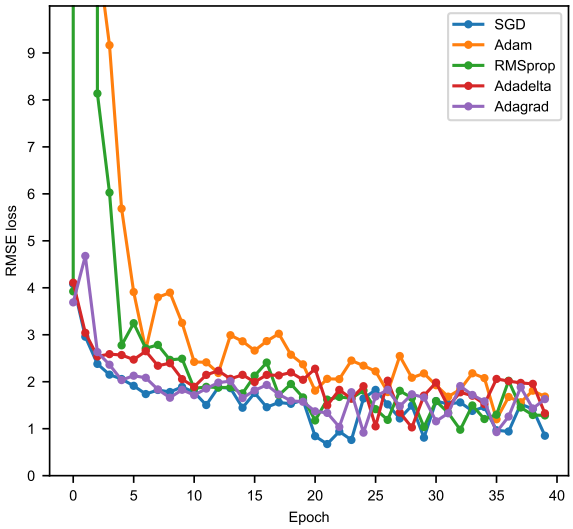

Fig. 7. RMSE losses of different optimizers on the ML dataset during training. The SGD optimizer converges at the lowest RMSE loss after 21 epochs, while other optimizers take longer iterations to converge.

popular optimization techniques. They are the Stochastic Gradient Descent (SGD), RMSProp [34], Adadelta [35], Adagrad [36], and Adam [37]. We observe that the training converges quite fast after about twenty epochs for certain optimizers such as SGD with momentum [38] or Adagrad. This greatly reduces the training time. Thus in both ML and BX datasets, we perform training with SGD with momentum value of 0.9. For the learning rate α, we choose two values, which are 0.001 and 0.0005. They both help us converge smoothly, but we find that the α rate of 0.001 performs slightly better than the α rate of 0.0005. Fig. 7 illustrates the converging loss of different optimizers.

VI. CONCLUSION

In this paper, we propose a novel framework called HeteGraph to handle heterogeneous graph-structured data to solve recommendation problems. The flexible architecture of HeteGraph enables the composition of different contextual models to learn high quality embeddings of the heterogeneous graph nodes and derive solution for various recommendation tasks. We present the important features of the framework including the bias neighbourhood sampling phase, the graph convolutional operation phase, the embedding objective and the application models. To evaluate how the graph convolutional operation technique can be used to solve recommendation problems, we propose two novel models for two different recommendation tasks: item rating prediction and diversified recommendations. We perform extensive evaluations on these models and our proposed methods achieve encouraging results. In the future work, we plan to improve our model in terms of accuracy measurement, as well as the sampling technique to account for more edge's attributes.

REFERENCES

[1] G. Adomavicius and A. Tuzhilin, "Toward the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions," *IEEE Trans. Knowl. Data Eng.*, vol. 17, no. 6, pp. 734–749, 2005.

[2] B. Hidasi and A. Karatzoglou, "Recurrent Neural Networks with Top-k Gains for Session-based Recommendations," in *Proc. of the 27th ACM International Conference on Information and Knowledge Management (CIKM 2018)*, Torino, Italy, October 2018, pp. 843–852.

[3] S. Sedhain, A. K. Menon, S. Sanner, and L. Xie, "AutoRec: Autoencoders Meet Collaborative Filtering," in *Proc. of the 24th International Conference on World Wide Web Companion Volume (WWW 2015)*, Florence, Italy, May 2015, pp. 111–112.

[4] W. Kang and J. McAuley, "Self-Attentive Sequential Recommendation," in *Proc. of the IEEE International Conference on Data Mining (ICDM 2018)*, Singapore, November 2018, pp. 197–206.

[5] W. L. Hamilton, R. Ying, and J. Leskovec, "Representation Learning on Graphs: Methods and Applications," *IEEE Data Eng. Bull.*, vol. 40, no. 3, pp. 52–74, 2017.

[6] T. N. Kipf and M. Welling, "Semi-Supervised Classification with Graph Convolutional Networks," in *Proc. of the 5th International Conference on Learning Representations (ICLR 2017)*, Toulon, France, April 2017.

[7] W. L. Hamilton, Z. Ying, and J. Leskovec, "Inductive Representation Learning on Large Graphs," in *Proc. of the 31st Annual Conference on Neural Information Processing Systems (NIPS 2017)*, Long Beach, CA, USA, December 2017, pp. 1025–1035.

[8] A. Y. Ng, M. I. Jordan, and Y. Weiss, "On Spectral Clustering: Analysis and an algorithm," in *Proc. of the 15th Annual Conference on Neural Information Processing Systems (NIPS 2001)*, Vancouver, British Columbia, Canada, December 2001, pp. 849–856.

[9] J. B. Kruskal, "Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis," *Psychometrika*, vol. 29, no. 1, pp. 1–27, Mar. 1964.

[10] B. Perozzi, R. Al-Rfou, and S. Skiena, "DeepWalk: online learning of social representations," in *Proc. of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2014)*, New York, NY, USA, August 2014, pp. 701–710.

[11] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, "LINE: Large-scale Information Network Embedding," in *Proc. of the 24th International Conference on World Wide Web (WWW 2015)*, Florence, Italy, May 2015, pp. 1067–1077.

[12] D. Wang, P. Cui, and W. Zhu, "Structural Deep Network Embedding," in *Proc. of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2016)*, San Francisco, CA, USA, August 2016, pp. 1225–1234.

[13] A. Grover and J. Leskovec, "Node2vec: Scalable Feature Learning for Networks," in *Proc. of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2016)*, San Francisco, CA, USA, August 2016, pp. 855–864.

[14] S. Cao, W. Lu, and Q. Xu, "GraRep: Learning Graph Representations with Global Structural Information," in *Proc.of the 24th ACM International Conference on Information and Knowledge Management (CIKM 2015)*, Melbourne, VIC, Australia, October 2015, pp. 891–900.

[15] Z. Yang, W. W. Cohen, and R. Salakhutdinov, "Revisiting Semi-Supervised Learning with Graph Embeddings," in *Proc. of the 33nd International Conference on Machine Learning (ICML 2016)*, New York City, NY, USA, June 2016, pp. 40–48.

[16] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral Networks and Locally Connected Networks on Graphs," in *Proc. of the 2nd International Conference on Learning Representations (ICLR 2014)*, Banff, AB, Canada, April 2014.

[17] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst, "Geometric Deep Learning: Going beyond Euclidean data," *IEEE Signal Process. Mag.*, vol. 34, no. 4, pp. 18–42, 2017.

[18] H. Dai, B. Dai, and L. Song, "Discriminative Embeddings of Latent Variable Models for Structured Data," in *Proc. of the 33nd International Conference on Machine Learning (ICML 2016)*, New York City, NY, USA, June 2016, pp. 2702–2711.

[19] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering," in *Proc. of the 30th Annual Conference on Neural Information Processing Systems (NIPS 2016)*, Barcelona, Spain, December 2016, pp. 3837–3845.

[20] D. K. Duvenaud, D. Maclaurin, J. Aguilera-Iparraguirre, R. Gómez-Bombarelli, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams, "Convolutional Networks on Graphs for Learning Molecular Fingerprints," in *Proc. of the 29th Annual Conference on Neural Information Processing Systems (NIPS 2015)*, Montreal, Quebec, Canada, December 2015, pp. 2224–2232.

[21] F. Monti, M. M. Bronstein, and X. Bresson, "Geometric Matrix Completion with Recurrent Multi-Graph Neural Networks," in *Proc. of the 31st Annual Conference on Neural Information Processing Systems (NIPS 2017)*, Long Beach, CA, USA, December 2017, pp. 3700–3710.

[22] M. Zitnik, M. Agrawal, and J. Leskovec, "Modeling polypharmacy side effects with graph convolutional networks," *Bioinformatics*, vol. 34, no. 13, pp. i457–i466, 2018.

[23] R. van den Berg, T. N. Kipf, and M. Welling, "Graph convolutional matrix completion," *CoRR*, vol. abs/1706.02263, 2017. [Online]. Available: http://arxiv.org/abs/1706.02263

[24] H. Cheng, L. Koc, J. Harmsen, T. Shaked, T. Chandra, H. Aradhye, G. Anderson, G. Corrado, W. Chai, M. Ispir, R. Anil, Z. Haque, L. Hong, V. Jain, X. Liu, and H. Shah, "Wide & Deep Learning for Recommender Systems," in *Proc. of the 1st Workshop on Deep Learning for Recommender Systems (DLRS@RecSys 2016)*, Boston, MA, USA, September 2016, pp. 7–10.

[25] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec, "Graph Convolutional Neural Networks for Web-Scale Recommender Systems," in *Proc. of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD 2018)*, London, UK, August 2018, pp. 974–983.

[26] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Proc. of the 27th Annual Conference on Neural Information Processing Systems (NIPS 2013)*, Lake Tahoe, Nevada, USA, December 2013, pp. 3111–3119.

[27] G. Linden, B. Smith, and J. York, "Amazon.com Recommendations: Item-to-Item Collaborative Filtering," *IEEE Internet Computing*, vol. 7, no. 1, pp. 76–80, 2003.

[28] C. A. Gomez-Uribe and N. Hunt, "The Netflix Recommender System: Algorithms, Business Value, and Innovation," *ACM Trans. Management Inf. Syst.*, vol. 6, no. 4, pp. 13:1–13:19, 2016.

[29] C. Ziegler, S. M. McNee, J. A. Konstan, and G. Lausen, "Improving recommendation lists through topic diversification," in *Proceedings of the 14th international conference on World Wide Web, WWW*, Chiba, Japan, May 2005, pp. 22–32.

[30] Y. Koren, R. M. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *IEEE Computer*, vol. 42, no. 8, pp. 30–37, 2009.

[31] X. Luo, M. Zhou, Y. Xia, and Q. Zhu, "An efficient non-negative matrix-factorization-based approach to collaborative filtering for recommender systems," *IEEE Trans. Industrial Informatics*, vol. 10, no. 2, pp. 1273–1284, 2014.

[32] D. Lemire and A. Maclachlan, "Slope one predictors for online rating-based collaborative filtering," in *Proceedings of the 2005 SIAM International Conference on Data Mining, SDM 2005*, Newport Beach, CA, USA, April 2005, pp. 471–475.

[33] B. M. Sarwar, G. Karypis, J. A. Konstan, and J. Riedl, "Item-based collaborative filtering recommendation algorithms," in *Proceedings of the Tenth International World Wide Web Conference, WWW 10*, May 2001, pp. 285–295.

[34] A. Graves, "Generating sequences with recurrent neural networks," *CoRR*, vol. abs/1308.0850, 2013. [Online]. Available: http://arxiv.org/abs/1308.0850

[35] M. D. Zeiler, "ADADELTA: an adaptive learning rate method," *CoRR*, vol. abs/1212.5701, 2012. [Online]. Available: http://arxiv.org/abs/1212.5701

[36] J. C. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *J. Mach. Learn. Res.*, vol. 12, pp. 2121–2159, 2011.

[37] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations, ICLR 2015, Conference Track Proceedings*, San Diego, CA, USA, May 2015.

[38] I. Sutskever, J. Martens, G. E. Dahl, and G. E. Hinton, "On the importance of initialization and momentum in deep learning," in *Proceedings of the 30th International Conference on Machine Learning, ICML 2013*, vol. 28, Atlanta, GA, USA, June 2013, pp. 1139–1147.