Neural Computing and Applications (2023) 35:13047–13063
https://doi.org/10.1007/s00521-020-05667-z

S.I. : DEEP SOCIAL COMPUTING

HeteGraph: graph learning in recommender systems via graph convolutional networks

Dai Hoang Tran[1] · Quan Z. Sheng[1] · Wei Emma Zhang[2] · Abdulwahab Aljubairy[1] · Munazza Zaib[1] · Salma Abdalla Hamad[1] · Nguyen H. Tran[3] · Nguyen Lu Dang Khoa[4]

Received: 2 June 2020 / Accepted: 27 December 2020 / Published online: 8 January 2021

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. 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 the encouraging performance of *HeteGraph* on the first task and the state-of-the-art performance on the second task.

Keywords Recommender systems · Graph convolutional network · Heterogeneous graphs · Neural networks

1 Introduction

Nowadays, online users are surrounded with huge amount of information, and having great difficulty when making decision from online services such as e-commerce, music and news. Recommender systems have been developed and adopted as effective solutions, where users are recommended items tailored to their needs and preferences. Due to their practicality, recommender systems have been an active research field until now. The early methods were mainly based on the principles of the neighbourhood collaborative filtering and content-based filtering [1]. Then, model-based methods became dominant thanks to their efficiency and accuracy, especially the matrix factorization approaches [2].

However, in recent years, we are seeing an increasing amount of deep learning methods that contribute greatly to this field. Several new approaches using deep learning methods have been applied and achieved promising results [3–6]. Particularly, the combination of deep learning methods and graph methodologies are commonly used to

✉ Dai Hoang Tran
dai-hoang.tran@hdr.mq.edu.au

Quan Z. Sheng
michael.sheng@mq.edu.au

Wei Emma Zhang
wei.e.zhang@adelaide.edu.au

Abdulwahab Aljubairy
abdulwahab.aljubairy@hdr.mq.edu.au

Munazza Zaib
munazza-zaib.ghori@students.mq.edu.au

Salma Abdalla Hamad
salma-abdalla-ibrahim-mah.h@students.mq.edu.au

Nguyen H. Tran
nguyen.tran@sydney.edu.au

Nguyen Lu Dang Khoa
Khoa.nguyen@data61.csiro.au

1 Macquarie University, Sydney, Australia
2 The University of Adelaide, Adelaide, Australia
3 The University of Sydney, Sydney, Australia
4 CSIRO Data61, Sydney, Australia

solve recommendation problems, thanks to their synergy. The common workflow of this combined approach is illustrated in Fig. 1. The complete workflow involves 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 in Fig. 1 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 the solving of 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. 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 [7].

One popular approach in deep learning that recently has spurred a lot of exciting developments is the graph convolutional networks (GCNs) [8]. GCNs are neural network models that have been designed to work with graph-structured data using graph adjacency matrix. Based on GCNs, GraphSage [9] 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 Fig. 2. 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. Additionally, each node or edge can have its own set of attributes such as movie genre and title or user age and location. 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 in this paper follows the new direction of using deep learning in solving recommendation problems. Particularly, we look at the problem of making recommendations as 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 our 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 recommendation tasks.

The first task is *item rating prediction*. It is about making a rating prediction between a user and an item, which is a fundamental task of a recommender system.

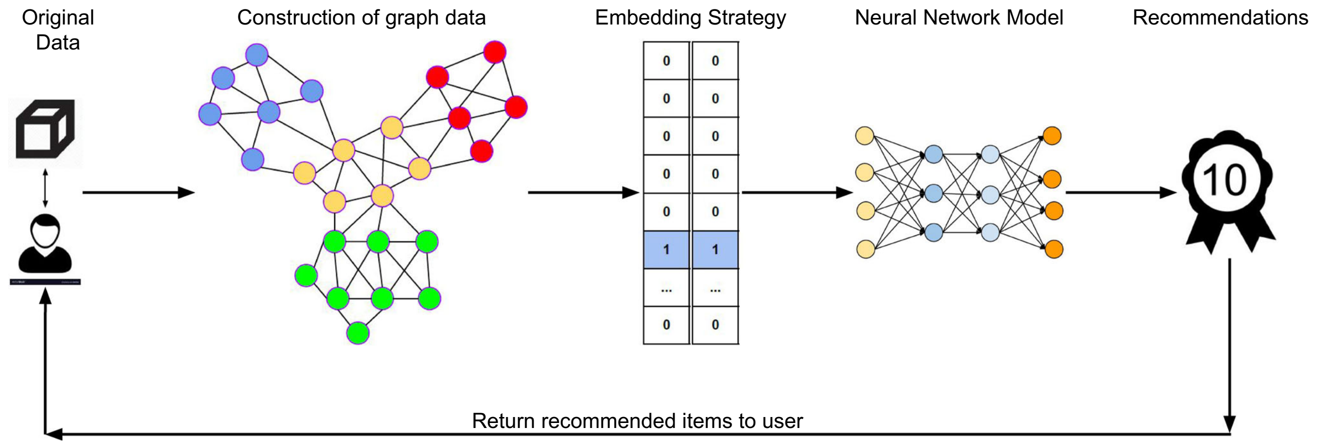

Fig. 1 Common workflow of deep learning recommender systems using graph-constructed data. Users, items and the interaction information are constructed as graph-structured data. Then, the embedding method will derive the useful embeddings of those data which will be fed into deep neural network model.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

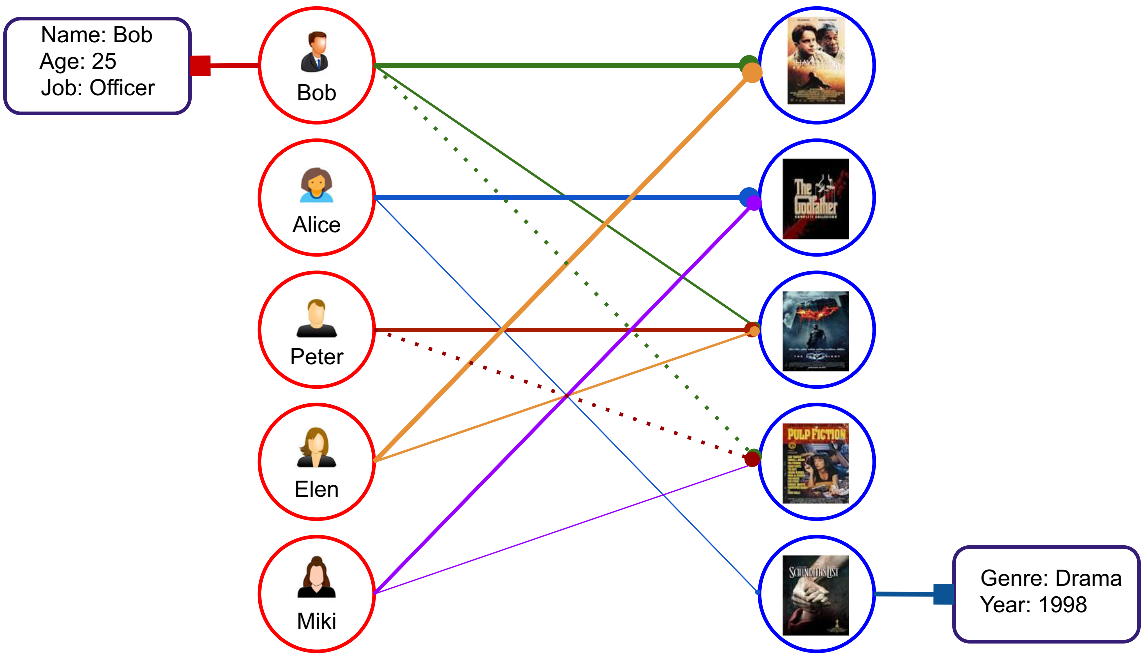

Fig. 2 Example of a heterogeneous graph in a movie recommender system. There are user node type and movie node type. Each node type has its own attributes. The edges (lines) between these nodes are their interactions which in this case is the rating that a user gives to a specific movie. The rating score is represented by the thickness of the edge. The dash line is the rating prediction that the system wants to perform

Specifically, we want to show that using GCO technique can help generate more accurate rating predictions based on user and item interactions. The second task is the ability to generate a diversified list of items to recommend to an active user. This second task, namely *diversified item recommendations*, 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 can 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 HeteGraph framework allows us to contribute several aspects in solving recommendation problems based on graph convolutional principles. Our main contributions of this work are as follows:

- An adapted graph convolutional operation from the GCN approaches to work with heterogeneous graph-structured data. This process includes the heterogeneous neighbourhood sampling and the adapted GCO to learn graph embeddings. These embeddings will then can be used for different recommendation applications.
- A novel framework named HeteGraph based on our adapted GCO to solve challenging recommendation tasks. We conduct experiments with two recommendation tasks that have different objectives as a proof of concept to show how HeteGraph can help solve recommendation problems.
- The strong performant evaluation on *item rating prediction* and *diversified item recommendations* tasks by using the HeteGraph framework on two real-world datasets.

This work is an extended version from our previous work accepted in the World Congress on Computational Intelligence Conference (WCCI 2020). Compared to the previous work, we provide a more in-depth model explanation, and elaborate more details in the evaluation process, including the model-parameters tuning procedures through our observation. The rest of this paper is organized as follows. In Sect. 2, we overview the related works on recent researches of GCNs and its applications in recommender systems. We detail the HeteGraph framework architecture in Sect. 3. We illustrate the recommendation application models in Sect. 4. The evaluations are described in Sect. 5, and we conclude our work in Sect. 6.

2 Related work

In recent years, the works on embedding learning of graph-structured data have gained significant attention. 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. We outline here the mainstream graph embedding approaches and how they are used in recommender systems.

2.1 Graph embedding approaches

Matrix factorization approaches In the early development, the classical methods to learn node embedding of graph-structured data are spectral clustering [10], PageRank [11] and multi-dimensional scaling [12]. Based on these early works, several new and enhanced methods for deriving node embedding using random-walk and matrix factorization have been proposed [13–17]. One limitation of these algorithms is the *transductive* inference, because they train node embedding for each individual node. As such, new nodes which did not appear during the training phase can have poor inference for certain machine learning tasks. Still, a notable exception is the algorithm Planetoid [18], which is a semi-supervised embedding approach that can produce *inductive* inference. The main difference of Planetoid from our framework is that we leverage the graph-structure data for inference, while Planetoid just uses the graph-structure data for regularization in the training phase.

Supervised learning approaches Aside from the matrix factorization approaches, supervised learning of the graph-structured data is another popular approach, such as the graph kernel methods [19] or recent neural network algorithms [20–22]. These early works of leveraging deep learning methods have inspired the development of recent advanced methods of GCNs. The main difference of those approaches from our work is the objective. We focus on learning node embedding for downstream machine learning tasks, while previous approaches put their focus on learning and classifying the whole graph.

Graph convolutional network approaches 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. [23], where the concept was developed based on special graph theory. Continuing this line of research, several other researchers proposed improvements and extensions of this spectral convolution [8, 9, 20, 24–29], 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 or drug-design tasks [27, 28]. However, a strong limitation of these spectral GCN approaches is the requirement of the entire graph Laplacian $\mathcal{L}$ during the training phase [8], which is a prohibitively expensive operation for large graphs. To address this issue, GraphSage [9] employed the random-walk technique to sample node’s neighbourhood and performs GCO on the target node and its sampled neighbourhood to train the model. This new approach significantly reduces the memory consumption, and also removes the burden of working with the whole graph Laplacian matrix. Our framework is inspired by the work of GraphSage, and we improve it to work with heterogeneous graph-structured data, that is suitable for recommendation tasks, while the work of GraphSage only deals with homogeneous graph. Additionally, by extending the handling of graph data into heterogeneous setting, we enable more machine learning applications to deploy with our framework in compare to the limited amount of machine learning tasks in GraphSage. Due to this extension, our work cannot be measured against GraphSage evaluation since GraphSage did not provide recommendation task measurement in the original work.

2.2 Recommender systems based on graph convolution

Recommender systems have been traditionally relied on collaborating filtering approaches such as K-nearest neighbour or matrix factorization [1]. The field was seeing a stable adaption in the e-commerce industry such as Amazon and Netflix services. 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 [4], recurrent neural network [3], and wide and deep model [30]. Given the heterogeneous and graph-structured data properties of the recommender systems, there is a strong synergy by using deep learning methods with graph algorithms to solve recommendation tasks.

Henceforth, it is expected 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 [29] or belong to a proprietary and specific service’s task [31]. 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.

3 HeteGraph architecture

Our framework architecture is illustrated in Fig. 3. HeteGraph comprises of four phases, each of them handle a specific operation. The whole sequential process can be described as follows. For each node in the graph, we collect its neighbourhood via random-walk technique and learn the node embedding using our adapted GCO technique (Phase 1 and 2). We can learn the node embeddings either in

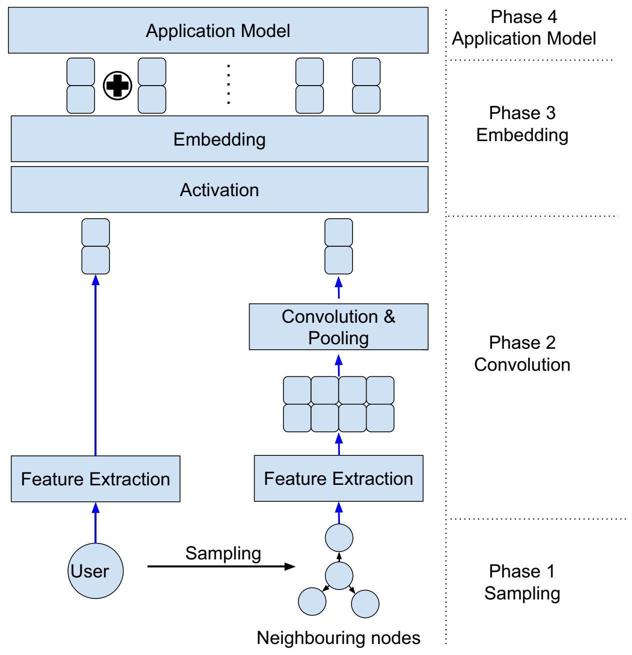

Fig. 3 The HeteGraph recommender framework for heterogeneous graph-structured data. In Phase 1, we sample the user neighbourhood. In Phase 2, we perform our GCO technique on the user and its neighbours features. In Phase 3, we learn the embeddings, and the embedding learning strategy depends on the application model of Phase 4

supervised or unsupervised manner depending on the specific application model (Phase 3). These embeddings are then used as inputs to solve the targeted recommendation task (Phase 4).

3.1 Phase 1—Node 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 to handle the graph heterogeneity. Figure 4 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. Due to the heterogeneity aspect, we need to use a 2k hop ($k \in N, k > 0$) so that each node can walk to other same-type nodes. For example, with $k = 1$, user A can reach user B via one common item I with 2-hop (both users have rated item I). Likewise, with $k = 2$, we can find connection between a user pairs or item pairs via 4-hop. By increasing k, we can sample each node neighbourhood with longer path, but at the expense of having lower relevancy between each connection, and longer sampling process.

Due to these drawbacks and with extensive experiments, we have decided to employ a 2-hop walking distance in our model ($k = 1$), since it gives the best performance in our algorithms based on the evaluations. The edge between a user-item node-pairs during the walk represents their interaction, and the edge weight represents that interaction affinity between a user-node and item-node. We leverage the edge weight to bias our walks when sampling. For instance, 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 orders. 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 $NB_p = \{p_1, p_2, \ldots, p_i\}$ and each path p_i contains heterogeneous nodes $p_i = \{n_1, n_2, \ldots, n_k\}$. The cardinality of $|NB_p|$ and $|p_i|$ is both predefined constants.

Algorithm 1 describes the implementation in pseudocode. 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).

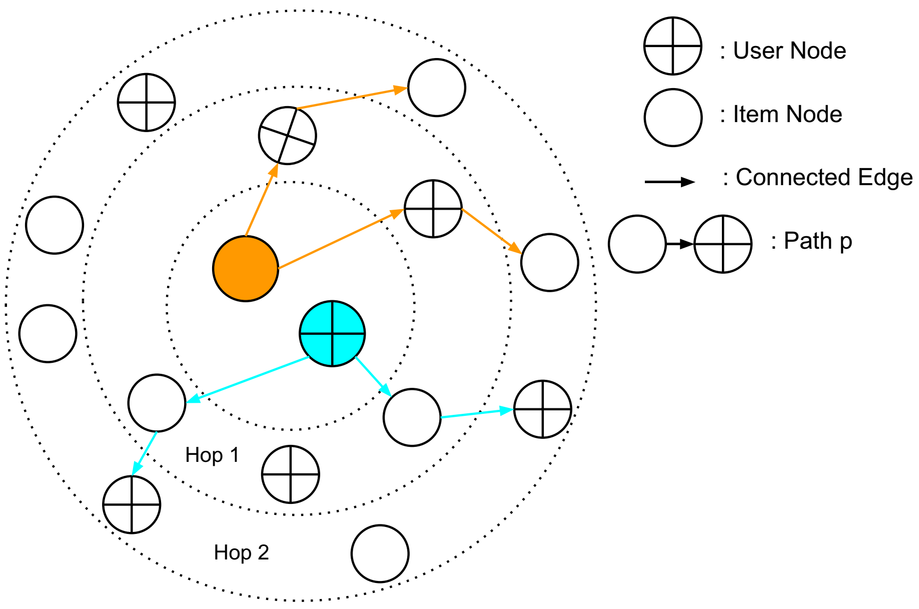

Fig. 4 HeteGraph neighbourhood sampling strategy. For a certain node of type χ, we perform a 2-hop random-walk to reach the next neighbour node of the same type χ. The walking path is biased by the connected edge's weight

```
Algorithm 1: HeteGraph Neighbourhood Sampling Algorithm (NeighSamp)
Input: node $n$ of Graph $\mathscr{G}(\mathscr{V},\mathscr{E})$; 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 $n$
begin
    $NB_n \longleftarrow \emptyset$
    for $i \leftarrow 1$ to $K$ do
        $target \leftarrow \emptyset$
        for $j \leftarrow 1$ to $D$ do
            if $j$ equal 1 then
                $target \leftarrow n$
            end
            $\{nb_j\} = NEIGHBOR(target)$
            $\{nb_j\}_{sort} \leftarrow SORT(\{nb_j\})$ in descending order
            $\{nb_j\}_{filter} \leftarrow$ select first $k+c$ elements from $\{nb_j\}_{sort}$
            $\{rand\_nb_j\} \leftarrow$ select $k$ elements from $RAND(\{nb_j\}_{filter})$
            $rnb_i \leftarrow RAND(\{rand\_nb_j\})$
            $NB_n[i]$ adds $rnb_i$
            $target \leftarrow rnb_i$
        end
    end
    return $NB_n$
end
```

Algorithm 2: HeteGraph Graph Convolutional Operation Algorithm

```
Input: Graph $\mathscr{G}(\mathscr{V},\mathscr{E})$; node attributes $\{x_n, \forall n \in V\}$; Pooling weight matrix $\mathbf{W}^{pool}$; Convolution
       weight matrix $\mathbf{W}$; non-linearity function $\sigma$; 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 $n \in \mathscr{V}$
1  begin
2      $Z \leftarrow \emptyset$
3      for $n \in \mathscr{V}$ do
4          $f_{NB_n} \leftarrow NEIGH\_AGG($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
```

3.2 Phase 2—Graph convolutional operation

3.2.1 Feature extraction

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. We use different techniques to transform raw data into vector inputs depend on the attribute data types. The detailed transformation is outlined as follows:

– *Text data type*
  Certain node attributes contain text description such as item's title, item's summary. To transform these textual attributes to vectorize form, we use pre-train word embedding such as [32] for each word, then we apply an GRU neural network [33] to translate them into a fixed size vector.
– *Category data type*
  Few node attributes contain categorical data type, such as user's location, and item's genre. To transform these categorical attributes to vectorize form, we use one-hot-encoding technique to transform them into a fixed size vector.
– *Numerical data type*
  For numerical data type attributes such as user's age, we apply the scaling transformation by using the standard score formulation as follows:

$$z = \frac{x - \mu}{\sigma} \tag{1}$$

  where μ is the mean of all samples, and σ is the standard deviation of all samples.
– *Rating scale*
  Finally, for the rating score as the interaction between a user node and item node, depending on the application models, we scale it differently. During the experimentation, we mostly use the explicit ratings value as the ground true values without any transformation. But when we perform ranking tasks which are not included in this paper, we transform them in implicit rating values that are zero and one.

With the transformation taking place, each node will be extracted as a featured vector in the graph. 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.

3.2.2 Convolutional operation

After each node gets its sampled neighbourhood, which is denoted by a set of bias random paths $NB_p = \{p_1, p_2, \ldots, p_i\}$, they become the inputs for our adapted GCO algorithm. This GCO process is critical because it extracts the prominent characteristics of a node's neighbourhood, and puts them into the node embedding. The main reason for doing this is to train our model to be able to infer node into node-embedding *transductively*, thanks to its local neighbourhood. Each node has its own neighbourhood, and similar nodes will have similar neighbour nodes. By training our model to be able to learn each node in conjunction with its neighbourhood properties, we allow to model to extract patterns of each node relationship with others regardless of node type or node location. As such, even when a new node is added to the graph, our model still can infer its embedding quickly without replying on the whole graph structure. This is why GCO process is vital for HeteGraph.

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

$$f_{NB_n} = \begin{bmatrix} att_{p_1} \\ att_{p_2} \\ \cdots \\ att_{p_i} \end{bmatrix}, \tag{2}$$

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" after an activation function:

$$f_{PO_n} = \text{mean}(\sigma(\mathbf{W}^{pool} \cdot f_{NB_n})), \tag{3}$$

$$\text{mean}\left(\begin{bmatrix} x_{11} & \ldots & x_{1m} \\ \vdots & \ddots & \vdots \\ x_{n1} & \ldots & x_{nm} \end{bmatrix}\right) = \begin{bmatrix} \frac{\sum_{i=1}^{n} x_{i1}}{n} & \ldots & \frac{\sum_{i=1}^{n} x_{im}}{n} \end{bmatrix} \tag{4}$$

The *attribute pooling* layer comes with its own weight parameters $\mathbf{W}^{pool}$ and performs the mean operation (4) 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 nonlinear 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).

3.3 Phase 3—Embedding learning strategy

The embedding phase is the next step in the process of learning node embedding of the heterogeneous graph. To learn the embedding process of each node during the training, we apply a particular loss method depending on the application model. The reasons for this decision are the strong connection of node embedding with its application. A rating prediction task has a different interpretation for node embedding than a ranking prediction task. Henceforth, in the HeteGraph framework, embedding phase and application phase are tightly coupling together. In this work, we employ two type of loss functions for two different embedding strategies. Firstly, we have a supervised model where we use the "root-mean-square error" (RMSE) loss function to train the model on every pair of user-item interactions.

$$L_{\text{RMSE}} = \sqrt{\frac{1}{n}\Sigma_{i=1}^{n}\left(r_i - y_i\right)^2}. \tag{5}$$

The RMSE value is also a standard measurement metric for the accuracy of recommendation problems based on explicit feedback. In Eq. (5), r_i and y_i are the predicted rating and the actual rating, respectively.

Secondly, 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-pairs (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-pairs (u, j), which has lower edge weight value (negative item).

$$L_{HL}(z_u z_i) = \mathbb{E}_{neg_u \sim P_{neg}(u)} \max\{0, z_u \cdot z_{neg_u} - z_u \cdot z_i + \Gamma\}. \tag{6}$$

Equation (6) 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. [34]. 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.

3.4 Phase 4—Application models

As mentioned in Sect. 3.3, 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 evaluate how effectively the GCO technique can be applied in solving recommendation problems. Additionally, these two tasks are also the canonical examples of recommendation tasks. The detail of our application models is explained in the next section.

4 Recommendation application models

We evaluate the HeteGraph framework by building a supervised model for task 1: *item rating predictions*, and an unsupervised model for task 2: *diversified item recommendations*. They represent different real-world scenarios of the recommender system's applications.

4.1 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 [35] or movie recommendations of Netflix [36]. In these scenarios, 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. Alternatively, when the system does not have an explicit rating mechanism, implicit rating mechanism can be deployed. Very often, we have e-commerce services that do not provide rating feedback to users. Thus, to record a user interaction with an item, implicit signals from the system can be used such as item clicks, item checkout, etc. These implicit signals can be encoded as rating 1, otherwise 0 for non-interactive items. The HeteGraph framework can handle both rating styles. Due to the choice of datasets, we use the explicit feedback scenario for our first model. But the other implicit rating scenario should work the same for any other recommendation datasets. 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-pairs, we want to predict whether this node-pairs should form an edge, based on the node-pairs' attributes and its neighbourhood. In case of link attribute inference, we also want to predict the weigh value of this edge if we use explicit rating feedback.

In this work, we build the first model for the rating prediction problem by using the feed-forward neural network multilayer perceptron (MLP). As depicted in Fig. 5, for every known rating between a user-item pair, 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} with the attention layer a_{u,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 two hidden layers for this MLP network during our training and evaluation steps. To train this model, we split each dataset into a training set and a test set with a ratio of 80% and 20%, respectively. Through Phase 1 to Phase 3 as explained in Sect. 3, we learn the edge embedding and use them as the model inputs, and we use the true explicit rating of that user-item node-pairs as the model 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 Attention Model ($HeteGraph_{at\_sup}$). This is the enhanced version in compare to our previous conference model $HeteGraph_{sup}$.

4.2 Model 2—Diversified item recommendations

While the first model concerns about the HeteGraph performance on the common task of a recommender system, which is rating prediction, we also would like to explore a new and less popular 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. Very often, diversity plays an important role in a recommender service. For instance, a travel service should recommend excited and unexplored local places for their users in a new travel destination since those are quite often the main motivation for many travellers. Another example where recommend diversified items can excel in increasing user satisfactory is the online video services. For example, researchers from YouTube show how improving diversity in their recommendations increases user engagement with their service [37].

Diversity concept can be quite opinionated depend on the applications. Thus in our application, we consider the diversity of items is based on the item attributes. A majority of 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. But 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.

As a naive approach, we can recommend random items to an active user. This could ensure our recommendations

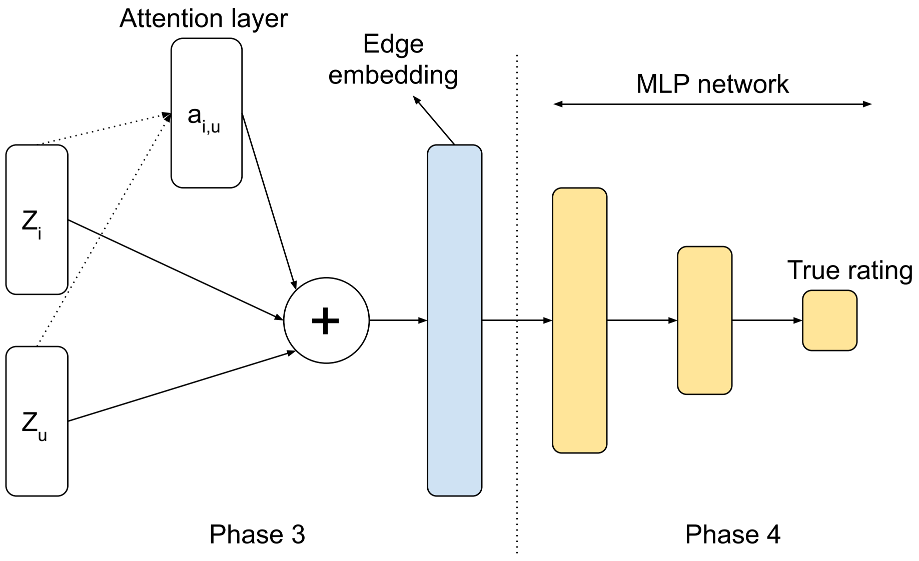

Fig. 5 HeteGraph Model 1—Item rating prediction model. In this model, we use attention layer with user and item embedding to form the edge embedding from the rating of a pair of user-item as input to the multilayer perceptron neural network (MLP) to train the rating prediction. The edge embedding is constructed via the concatenation operation C between the convolved tensor z_u of user u and the convolved tensor z_i of item i

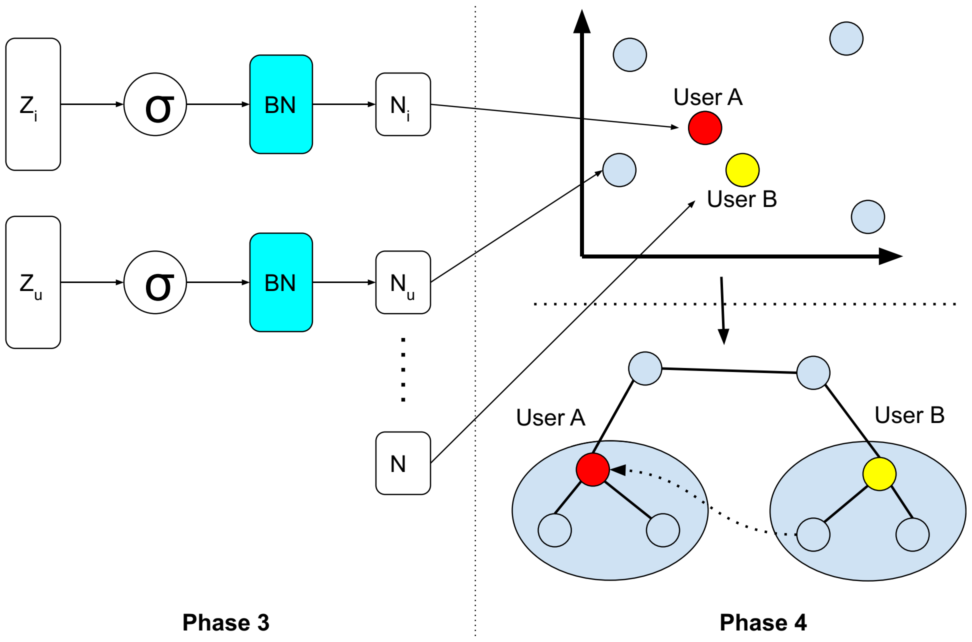

Fig. 6 HeteGraph Model 2—Long distance search for diversified recommendations. In this model, user-A and user-B have node embeddings that are close to each other in the embedding space, thus they can have similar characteristics and preferences. However, in the original graph, they belong to different 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

have huge diversity, but it would not capture user interest. Hence, a better strategy is to recommend new items and still preserve a certain level of relevance to that user's preferences. To be able to do this, we need to capture user-item interest score from her interaction history as well as the item novelty score based on the user interest level. Our strategy to accomplish this task is to build a second model named *HeteGraph*$_{bat\_div}$ as depicted in Fig. 6. It is clear in Fig. 6 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 implies 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. 6 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. In this enhanced model in compare to our conference model *HeteGraph*$_{div}$, we apply batch normalization during the training to improve the embedding learning accuracy.

5 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.

5.1 Preparation

We detail our dataset preprocessing steps, the evaluated metrics and compared algorithms in this section.

5.1.1 Datasets

We perform evaluations on three datasets, the "MovieLens 100K" (ML-100K), "MovieLens 1M" (ML-1M)[1] and "BookCrossing" (BX)[2]. The ML-100K 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. Similarly, the ML-1M is the bigger version of ML-100K with 1,000,000 ratings of 6,000 users and 4,000 movies. The rating density of ML-100K is 6.3% and 4.17% for ML-1M.

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 1 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. Table 1 shows the datasets statistic for our HeteGraph evaluation.

Table 1 Datasets for the evaluation

Dataset	ML-100K	ML-1M	BX-200K
User size	943	6,000	5,102
Item size	1,682	4,000	4,405
Rating size	100,000	1,000,000	219,289
Rating scale	1–5	1–5	0–10
Rating density	6.3%	4.17%	0.98%

5.1.2 Feature preprocessing

The ML datasets and the BX dataset come with little attribute information for item node type. For instance, the ML dataset provides movie's title, its online address, released date and genre, while the BX dataset provides book's title, its author, the publication date and image address. To be effective in capture semantic meaning of these items, we need more detailed information. As a result, we perform additional feature preprocessing to enrich the dataset attributes. This step also help our model phase two, when it does feature extraction before the GCO operation to have richer data to learn the hidden patterns. By using movies services such as IMDb[3] and TIMDb[4], we are able to collect a handful more attribute for movie item. Likewise, we apply the same procedure for the BX dataset using Amazon Book[5] to get additional book details such as description, category, etc. Table 2 lists the additional features we put into the ML and BX datasets for our HeteGraph platform. Adding these rich semantic features help us learn the node embedding accurately and effectively.

5.1.3 Evaluation metrics

We evaluate the HeteGraph framework performance on different metrics to see how effective our models are for a particular recommendation task. We describe these metrics below. These are off-line measurement metrics. It means they cannot be used to assert the true user's satisfaction upon receiving the recommendations.

- Intra-list Similarity (ILS) [38]: ILS measures the diversity of a recommendation list, and lower score means more diversity. Equation (7) 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.

[1] https://grouplens.org/datasets/movielens.
[2] https://grouplens.org/datasets/book-crossing.
[3] https://www.imdb.com.
[4] https://www.themoviedb.org.
[5] https://www.amazon.com.

Table 2 Additional attributes added to ML and BX datasets

Attributes	Type	Description	ML	BX
Summary	Text	A short summary about the content of the item. The summary usually contains five to ten sentences.	Yes	No
Keyword	Text	The main keywords that describes the item. Each keyword contains one to four words, describing different aspects of the item.	Yes	No
Review	Text	The user reviews of the item. If an item has more than twenty reviews, we choose random twenty of them, and ignore the rest. These reviews express the general sentiment of the item.	Yes	Yes
Language	Category	The spoken or written language of the item. The majority is English, but there are other languages as well.	Yes	Yes
Plot	Text	The main plot of the item, only available for the BX dataset. The plot is usually shorter than the summary.	No	Yes
Actor	Text	The actors that featured in the item. Only available for the ML dataset. One item can have multiple actors.	Yes	No
Author	Text	The author of the item. Only available for the BX dataset.	No	Yes

$$\mathrm{ILS}_u = \frac{\sum_{i_j \in L} \sum_{i_k \in L} sim(i_j, i_k)}{2|L|} \tag{7}$$

- Mean Absolute Error (MAE): MAE value evaluates the statistical difference between predicted ratings and true ratings for a recommendation list R. In Eq. (8), r_{ui} is the predicted rating and y_{ui} is the actual rating between the user u and item i. MAE is also less sensitive to outliers.

$$\mathrm{MAE} = \frac{1}{|R|} \sum_{r_{ui} \in R} |r_{ui} - y_{ui}| \tag{8}$$

- Root-Mean-Square Error (RMSE): RMSE is the most common accuracy measurement metric for recommendation task. Its formula is expressed in Eq. (5). Similar to MAE, RMSE represents the accuracy difference between predicted and actual ratings, but it is more sensitive to outliers.
- Precision-at-K (Pr@k) and Recall-at-K (Re@k): Both Pr@k and Re@k are metrics from the information retrieval research area. They are decision support metrics and do not consider the deviation between predicted ratings and actual ratings. They only measure how *relevant* a recommendation list to an active user. Eqs. (9) and (10) express the formulas. For a user u who receives a recommendation list p_u of size k, each item i in p_u is relevant to the user u if the relevant indicator function rel_{uj}'s value is 1. The rel_{ui} value is determined by the actual relevant list γ_u. Thus, the Pr@k value shows the probability that a predicted item is relevant in k items of p_u, while the Re@k value shows the probability that a relevant item is recommended in k items of p_u. The average Pr@k and Re@k scores are the average Pr_u@k scores and Re_u@k scores of all users in the test set. In out experiments, we set k to the value of 20.

$$Pr_u@k = \frac{\sum_{i=1}^{\min\{k,p_u\}} rel_{ui}}{k} \tag{9}$$

$$Re_u@k = \frac{\sum_{i=1}^{\min\{\mathrm{k},\mathrm{p_u}\}} rel_{ui}}{\gamma_u} \tag{10}$$

5.1.4 Comparison algorithms

To benchmark the recommendation tasks with the Hete-Graph framework, we compare our empirical results with five other algorithms: SVD [2], SVD++ [2], SlopeOne [39], NMF [40], and the kNN [41]. SVD and SVD++ are popular matrix factorization methods for recommender systems. SVD++ also accounts for implicit ratings while SVD does not. NMF is another matrix factorization algorithm that is similar to SVD, but it keeps the user and item factors positive. SlopeOne is a non-trivial item-based collaborative filtering algorithm, which has wide practical usages in real-world due to its simple and intuitive formulation. kNN is the basic user-based collaborative filtering method. The reasons we choose these algorithms for comparison with our models are that not only they are well known and being used in many e-commerce services [35, 36], but also we can reimplement these algorithms in our system. This gives us the confidence in our evaluated measurements. Admittedly, there are recent algorithms which leverage deep learning in recommender systems similar to our models. Unfortunately, we cannot implement those new methods in our system to compare with our models, due to lacking of the source-code as well as the datasets that they used.

5.2 Model evaluations

5.2.1 Accuracy evaluation 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 [38]. However, it is still the most common evaluation metric used nowadays to assert certain confidence on the capability of a recommender system. Hence, we measure our models performance based on four metrics. They are RMSE, MAE, Pr@K and Re@K as detailed in Sect. 5.1.3. We compare our model with other algorithms as described in Sect. 5.1.4. 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 and ML-1M dataset, we set k as 20, and the threshold score β as 3.5. For the BX dataset, we set k also as 20, and the threshold score β as 6. Table 3 shows our evaluation results. Surprisingly, both of the HeteGraph models achieve encouraging results. Admittedly, both of them cannot outscore the SVD or SVD++ in terms of RMSE and MAE metrics (lower is better), but we score higher than the SVD++ in the BX dataset for the Pr@k and Re@k metrics (higher is better).

Additionally, our models consistently perform better than the kNN, NMF and SlopeOne in the RMSE and MAE metrics in the ML dataset. For the comparison between our models variances HeteGraph$_{at\_sup}$ and HeteGraph$_{sup}$, it is clear that HeteGraph$_{at\_sup}$ 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 learn useful embedding. Additionally, by benchmarking the models on both ML-100K and ML-1M, we confirm the generalization of our models on the ML dataset. It is clear that the models perform better on ML-100K, but it keep being consistent in the raking with other methods when performing in ML-1M, this proves that the HeteGraph can generalize well on bigger dataset.

Table 3 Item rating prediction evaluation of application model 1

Metrics	RMSE	MAE	Pr@20	Re@20
ML-100K dataset				
HeteGraph$_{\mathbf{at\_sup}}$	**0.955**	**0.733**	**0.747**	**0.681**
HeteGraph$_{sup}$	0.976	0.761	0.737	0.679
SVD	0.943	0.743	0.763	0.659
SVD++	0.917	0.718	0.749	0.670
NMF	0.963	0.763	0.728	0.637
SlopeOne	0.950	0.745	0.733	0.654
kNN	0.981	0.776	0.717	0.706
ML-1M dataset				
HeteGraph$_{\mathbf{at\_sup}}$	**1.269**	**1.126**	**1.218**	**1.227**
HeteGraph$_{sup}$	1.379	1.231	0.997	1.134
SVD	1.267	1.199	0.987	1.089
SVD++	1.223	1.186	0.984	1.086
NMF	1.421	1.301	1.078	1.152
SlopeOne	1.391	1.332	1.043	1.161
kNN	1.435	1.387	1.227	1.239
BX-200K dataset				
HeteGraph$_{\mathbf{at\_sup}}$	**3.534**	**2.709**	**0.903**	**0.415**
HeteGraph$_{sup}$	3.702	2.802	0.843	0.388
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

5.2.2 Diversity evaluation of diversified recommendations (Model 2)

To verify the diversity of the recommendations, we use the intra-list similarity (ILS) score [38] 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

Table 4 Diversity Evaluation with ILS scores of Application Model 2

Dataset	ML-100K	ML-1M	BX-200K
HeteGraph$_{\mathbf{bat\_div}}$	**0.855**	**1.056**	**0.807**
HeteGraph$_{div}$	0.891	1.023	0.812
SVD	0.921	1.176	0.870
SVD++	0.914	1.135	0.893
SlopeOne	0.937	1.198	0.915
NMF	0.908	1.202	0.902
k-NN	0.945	1.338	0.921

that user's recommendation list size. Table 4 shows the evaluation results of the HeteGraph$_{bat\_div}$ and HeteGraph$_{div}$ (conference method) models. The enhanced model HeteGraph$_{bat\_div}$ with batch normalization show an increased score over the previous conference method. 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. To verify the results, we generate samples of recommendation list, and with the application is able to generate movies in different languages as well as variation in the genres. This is to ensure that an active user does not keep seeing recommended movies that is quite repetitive to her taste. We see similar results for BX dataset as well.

5.3 Hyper-parameters analysis and observations

Hyper-parameter tuning is vital in training a deep neural network. During the evaluations, we conduct experiments with different learning rate α to find the best learning rate value for our models. We also tune our models using different optimizers and regulation techniques. In this section, we briefly discuss our observations.

5.3.1 Hyper-parameters grid search

We make a grid-search attempt with different learning rate α, several optimizers and regularization techniques. We select five popular optimization techniques. They are the Stochastic Gradient Descent (SGD), RMSProp [42], Adadelta [43], Adagrad [44], and Adam [45]. We observe that the training converges quite fast after about twenty-one epochs for certain optimizers such as SGD with momentum [46] 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. Figure 7 illustrates the converging loss of different optimizers. We also see similar patterns when we apply these hyper-parameters to the ML-1M and BX-200K datasets. As a result, for all models, we decide to use SGD optimizer, learning rate of 0.001, and batch size of 256.

5.3.2 Random & sequential sampling of mini-batches

Another interesting observation during our evaluation is the effectiveness of mini-batch sampling strategy on different HeteGraph's models. Mini-batch gradient descent is a variation of the gradient descent method, where the training set data is split into small batches to calculate error loss and update the model's parameters. This is a very effective way to balance between the model's training speed as well as the model's learning capability.

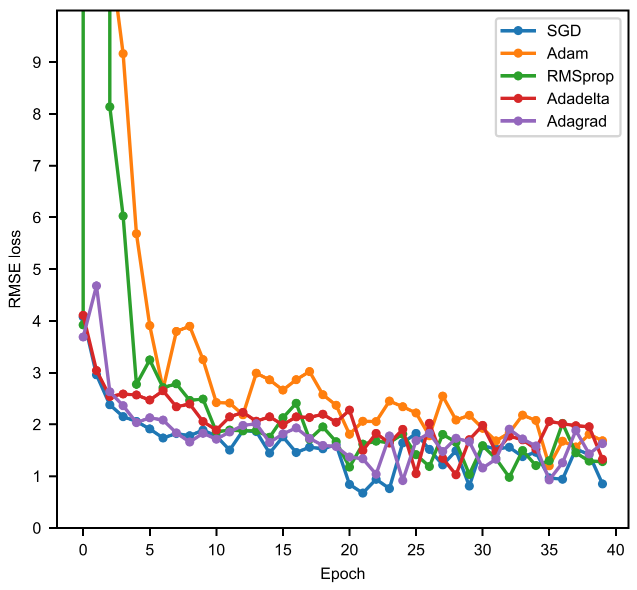

Fig. 7 RMSE losses of different optimizers on the ML-100K dataset during training. The SGD converges at the lowest RMSE loss after 21 epochs, while the other optimizers converge slower than the SGD

Normally, the data of every mini-batch is sampled randomly from the whole training set. However, we observe that using this random sampling strategy decreases our models' performance. To understand why, we look at the scores' distribution histogram of both datasets, and observe that each of them exhibits certain bias in their distribution. Figure 8 clearly depicts this issue. In the BX dataset, most of the rating mass is on the 0 rating score (which means implicit feedback), while in the ML dataset, most of the rating values are 3 or 4. Henceforth, to make our models learn better, instead of using random sampling, we apply sequential sampling for the mini-batches. This helps us derive more accurate models for the item rating prediction. Figure 9 illustrates the histogram of different optimizers for HeteGraph$_{mix}$ on the ML dataset. The x-axis represents the predicted scores during training, while the y-axis represents the amount of repetitions for a certain precondition-score, and get scaled down to the range of 0 to 1. As it can be seen that many of these histograms have similar structure to the data distribution of ML dataset in Fig. 8. It indicates that our sampling strategy makes the prediction scores to have a similar distribution in compare with the labels, thus increase the prediction accuracy of the models.

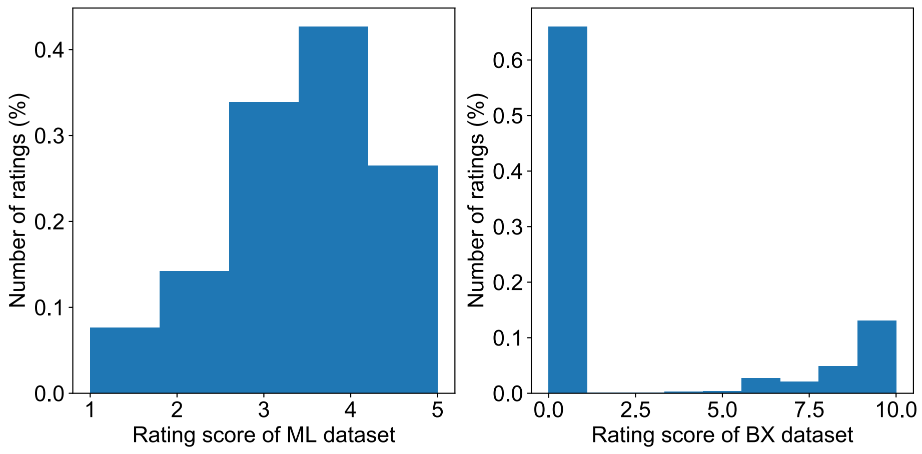

Fig. 8 Rating scores histogram of ML and BX datasets. The ML dataset has high percentage of score rating at value 3 and 4, while the BX dataset has most of the rating mass at value 0. They both have a skewed distribution

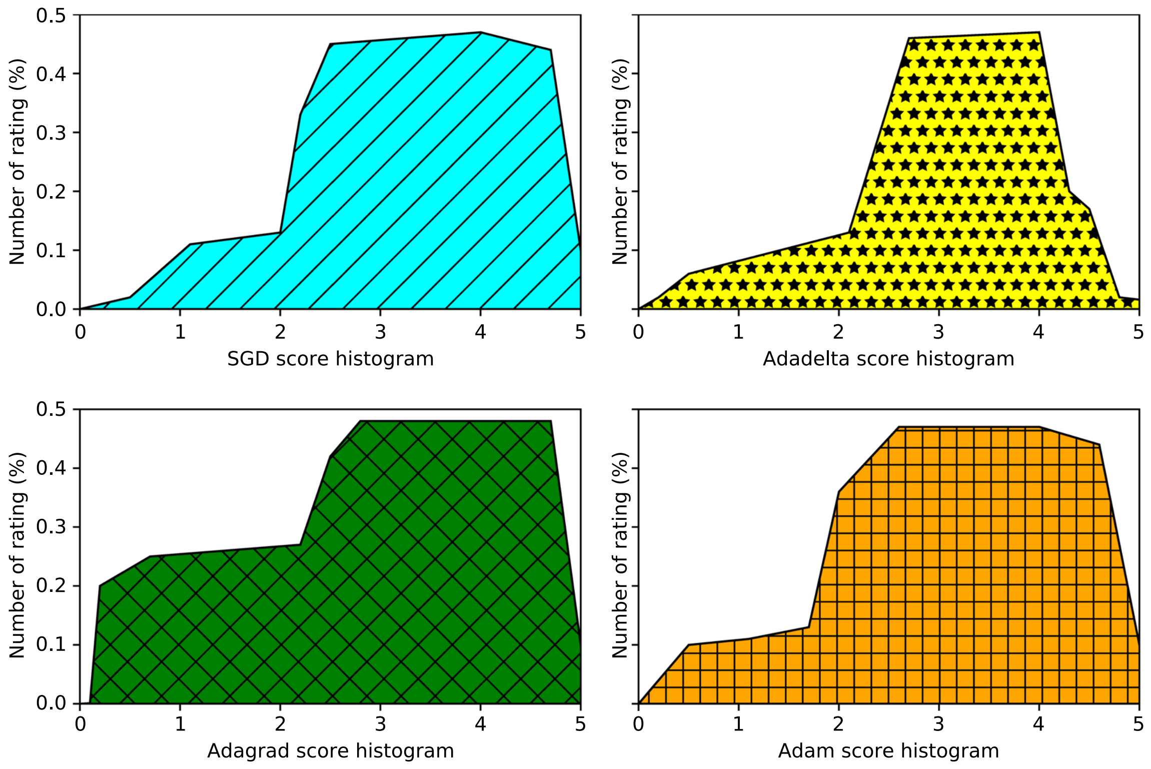

Fig. 9 Inference rating scores histograms of the ML dataset from different optimizers. Thanks to the sequential sampling, the histogram's shapes of these optimizers show similar structure to the ground-truth histogram of ML dataset in Fig. 8

6 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 GCO (graph convolutional operation) technique can be used to solve recommendation problems, we propose 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 the HeteGraph in two directions. First, we want to improve the neighbourhood sampling technique to account for more entities in the heterogeneous graph, since our current technique is only apply to two entities which are user and item. Second, we want to incorporate external knowledge graph sources such as Wikidata[6] into the framework to handle more machine learning tasks not just recommender systems.

Funding This study was funded by Australian Research Council Discovery Project ARC, Grant number DP200102298.

Availability of data and material Public dataset MovieLens is available at: https://grouplens.org/datasets/movielensPublic dataset Book-Crossing is available at: www2.informatik.uni-freiburg.de/~cziegler/BX.

Code availability The source code of HeteGraph is available at: http://github.com/heroddaji/dai_hetegraph.

Compliance with ethical standards

Conflict of Interest Author Quan Z. Sheng has received research grants from Company Australian Research Council.

References

1. Adomavicius G, Tuzhilin A (2005) Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions. IEEE Trans Knowl Data Eng 17(6):734–749
2. Koren Y, Bell RM, Volinsky C (2009) Matrix factorization techniques for recommender systems. IEEE Comput 42(8):30–37
3. Hidasi B, Karatzoglou A (2018) Recurrent neural networks with Top-k gains for session-based recommendations. In: Proceedings of the 27th ACM international conference on information and knowledge management (CIKM 2018), Torino, Italy, pp 843–852
4. Sedhain S, Menon AK, Sanner S, Xie L (2015) AutoRec: autoencoders meet collaborative filtering. In: Proceedings of the 24th international conference on world wide web companion volume (WWW 2015), Florence, Italy, pp 111–112
5. Kang W-C, McAuley J (2018) Self-attentive sequential recommendation. In: Proceedings of the IEEE international conference on data mining (ICDM 2018), pp 197–206, Singapore
6. Liu F, Xue S, Wu J, Zhou C, Hu W, Paris C, Nepal S, Yang J, Yu PS (2020) Deep learning for community detection: Progress, challenges and opportunities. In: Proceedings of the twenty-ninth international joint conference on artificial intelligence, IJCAI-20, pp 4981–4987
7. Hamilton WL, Ying R, Leskovec J (2017) Representation learning on graphs: methods and applications. IEEE Data Eng Bull 40(3):52–74
8. Kipf TN, Welling M (2017) Semi-supervised classification with graph convolutional networks. In: Proceedings of the 5th international conference on learning representations (ICLR 2017), Toulon, France
9. Hamilton WL, Ying Z, Leskovec J (2017) Inductive representation learning on large graphs. In: Proceedings of the 31st annual conference on neural information processing systems (NIPS 2017), Long Beach, CA, USA, pp 1025–1035
10. Ng AY, Jordan MI, Weiss Y (2001) On spectral clustering: analysis and an algorithm. In: Proceedings of the 15th annual conference on neural information processing systems (NIPS 2001), Vancouver, British Columbia, Canada, pp 849–856
11. Page L, Brin S, Motwani R, Winograd T (1999) The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab
12. Kruskal JB (1964) Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. Psychometrika 29(1):1–27
13. Perozzi B, Al-Rfou R, Skiena S (2014) DeepWalk: online learning of social representations. In: Proceedings of the 20th ACM SIGKDD international conference on knowledge discovery and data mining (KDD 2014), New York, NY, USA, pp 701–710
14. Tang J, Qu M, Wang M, Zhang M, Yan J, Mei Q (2015) LINE: large-scale information network embedding. In: Proceedings of the 24th international conference on world wide web (WWW 2015), Florence, Italy, pp 1067–1077
15. Wang D, Cui P, Zhu W (2016) Structural Deep Network Embedding. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2016), San Francisco, CA, USA, pp 1225–1234
16. Grover A, Leskovec J (2016) Node2vec: Scalable Feature Learning for Networks. In: Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining (KDD 2016), San Francisco, CA, USA, pp 855–864
17. Cao S, Lu W, Xu Q (2015) GraRep: Learning Graph Representations with Global Structural Information. In: Proceedings of the 24th ACM international conference on information and knowledge management (CIKM 2015), Melbourne, VIC, Australia, pp 891–900
18. Yang Z, Cohen WW, Salakhutdinov R (2016) Revisiting Semi-Supervised Learning with Graph Embeddings. In: Proceedings of the 33nd international conference on machine learning (ICML 2016), New York City, NY, USA, pp 40–48
19. Shervashidze N, Schweitzer P, van Leeuwen EJ, Mehlhorn K, Borgwardt KM (2011) Weisfeiler-lehman graph kernels. J Mach Learn Res 12:2539–2561
20. Dai H, Dai B, Song L (2016) Discriminative embeddings of latent variable models for structured data. In: Proceedings of the 33nd international conference on machine learning (ICML 2016), New York City, NY, USA, pp 2702–2711
21. Beck D, Haffari G, Cohn T (2018) Graph-to-Sequence Learning using Gated Graph Neural Networks. In: Proceedings of the 56th annual meeting of the association for computational linguistics (ACL 2018), Melbourne, Australia, pp 273–283
22. Scarselli F, Gori M, Tsoi AC, Markus H, Gabriele M (2009) The Graph Neural Network Model. IEEE Trans Neural Netw 20(1):61–80
23. Bruna J, Zaremba W, Szlam A, LeCun Y (2014) Spectral networks and locally connected networks on graphs. In: Proceedings of the 2nd international conference on learning representations (ICLR 2014), Banff, AB, Canada
24. Bronstein MM, Bruna J, LeCun Y, Szlam A, Vandergheynst P (2017) Geometric deep learning: going beyond Euclidean data. IEEE Signal Process Mag 34(4):18–42
25. Defferrard M, Bresson X, Vandergheynst P (2016) Convolutional neural networks on graphs with fast localized spectral filtering. In: Proceedings of the 30th annual conference on neural information processing systems (NIPS 2016), Barcelona, Spain, pp 3837–3845
26. Duvenaud DK, Maclaurin D, Aguilera-Iparraguirre J, Gómez-Bombarelli R, Hirzel T, Aspuru-Guzik A, Adams RP (2015) Convolutional Networks on Graphs for Learning Molecular Fingerprints. In: Proceedings of the 29th annual conference on

[6] https://www.wikidata.org.

neural information processing systems (NIPS 2015), Montreal, Quebec, Canada, pp 2224–2232
27. Monti F, Bronstein MM, Bresson X (2017) Geometric Matrix Completion with Recurrent Multi-Graph Neural Networks. In: Proceedings of the 31st Annual Conference on Neural Information Processing Systems (NIPS 2017), pages 3700–3710, Long Beach, CA, USA
28. Zitnik M, Agrawal M, Leskovec J (2018) Modeling polypharmacy side effects with graph convolutional networks. Bioinformatics 34(13):i457–i466
29. van den Berg R, Kipf TN, Welling M (2017) Graph convolutional matrix completion. CoRR, arXiv:abs/1706.02263
30. Cheng H-T, Koc L, Harmsen J, Shaked T, Chandra T, Aradhye H, Anderson G, Corrado G, Chai W, Ispir M, Anil R, Haque Z, Hong L, Jain V, Liu X, Shah H (September 2016) Wide & Deep Learning for Recommender Systems. In: Proceedings of the 1st Workshop on Deep Learning for Recommender Systems (DLRS@RecSys 2016), Boston, MA, USA, pp 7–10
31. Ying R, He R, Chen K, Eksombatchai P, Hamilton WL, Leskovec J (August 2018) Graph Convolutional Neural Networks for Web-Scale Recommender Systems. In: Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining (KDD 2018), London, UK, pp 974–983
32. Pennington J, Socher R, Christopher D (2014) Manning. Glove: Global vectors for word representation. In: EMNLP 2014, ACL, pp 1532–1543
33. Cho K, van Merrienboer B, Gülçehre Ç, Bahdanau D, Bougares F, Schwenk H, Bengio Y (2014) Learning phrase representations using RNN encoder-decoder for statistical machine translation. EMNLP 2014:1724–1734
34. Mikolov T, Sutskever I, Chen K, Corrado GS, Dean J (2013) Distributed representations of words and phrases and their compositionality. In: Proceedings of the 27th annual conference on neural information processing systems (NIPS 2013), pp 3111–3119, Lake Tahoe, Nevada, USA
35. Linden G, Smith B, York J (2003) Amazon.com Recommendations: Item-to-Item Collaborative Filtering. IEEE Internet Comput 7(1):76–80
36. Gomez-Uribe CA, Hunt N (2016) The netflix recommender system: algorithms, business value, and innovation. ACM Trans Manag Inf Syst 6(4):13:1–13:19
37. Wilhelm M, Ramanathan A, Bonomo A, Jain S, Chi EH, Jennifer G (2018) Practical diversified recommendations on youtube with determinantal point processes. In: Proceedings of the 27th ACM international conference on information and knowledge management, CIKM 2018, Torino, Italy, October 22-26, 2018, pp 2165–2173. ACM
38. Ziegler C-N, McNee SM, Konstan JA, Lausen G (2005) Improving recommendation lists through topic diversification. In: Proceedings of the 14th international conference on world wide web, WWW, Chiba, Japan, pp 22–32
39. Lemire D, Maclachlan A (2005) 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, pp 471–475
40. Luo X, Zhou M, Xia Y, Zhu Q (2014) An efficient non-negative matrix-factorization-based approach to collaborative filtering for recommender systems. IEEE Trans Ind Inform 10(2):1273–1284
41. Sarwar BM, Karypis G, Konstan JA, Riedl J (2001) Item-based collaborative filtering recommendation algorithms. In: Proceedings of the tenth international world wide web conference, WWW 10, pp 285–295
42. Graves A (2013) Generating sequences with recurrent neural networks. CoRR, arXiv:abs/1308.0850
43. Zeiler MD (2012) ADADELTA: an adaptive learning rate method. CoRR, arXiv:abs/1212.5701
44. Duchi JC, Hazan E, Singer Y (2011) Adaptive subgradient methods for online learning and stochastic optimization. J Mach Learn Res 12:2121–2159
45. Kingma DP, Ba J (2015) Adam: A method for stochastic optimization. In: 3rd International conference on learning representations, ICLR conference track proceedings. San Diego, CA, USA, (May 2015)
46. Sutskever I, Martens J, Dahl GE, Hinton GE (2013) 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, pp 1139–1147