STAR-GCN: Stacked and Reconstructed Graph Convolutional Networks for Recommender Systems

Jiani Zhang[1] , **Xingjian Shi**[2] , **Shenglin Zhao**[3] and **Irwin King**[1]
[1] The Chinese University of Hong Kong, Hong Kong, China
[2]Hong Kong University of Science and Technology, Hong Kong, China
[3]Youtu Lab, Tencent, Shenzhen, China
{jnzhang, king}@cse.cuhk.edu.hk, xshiab@connect.ust.hk, zsl.zju@gmail.com

Abstract

We propose a new *STAcked and Reconstructed Graph Convolutional Networks* (STAR-GCN) architecture to learn node representations for boosting the performance in recommender systems, especially in the cold start scenario. STAR-GCN employs a stack of GCN encoder-decoders combined with intermediate supervision to improve the final prediction performance. Unlike the graph convolutional matrix completion model with one-hot encoding node inputs, our STAR-GCN learns low-dimensional user and item latent factors as the input to restrain the model space complexity. Moreover, our STAR-GCN can produce node embeddings for new nodes by reconstructing masked input node embeddings, which essentially tackles the cold start problem. Furthermore, we discover a label leakage issue when training GCN-based models for link prediction tasks and propose a training strategy to avoid the issue. Empirical results on multiple rating prediction benchmarks demonstrate our model achieves state-of-the-art performance in four out of five real-world datasets and significant improvements in predicting ratings in the cold start scenario. The code implementation is available in https://github.com/jennyzhang0215/STAR-GCN.

1 Introduction

Recommender systems, which analyze users' preference patterns to suggest potential targets, are indispensable in content providers, electronic retailers, web search engines, etc. The key mathematical problem underlying recommender systems is matrix completion [Candès and Recht, 2009]. Assume there are N users and M items, the recommendation algorithm aims to fill in the missing entries in the $N \times M$ rating matrix given the existing entries.

The classical way to solve this problem is via *Matrix Factorization* (MF) [Koren *et al.*, 2009], in which the rating scores are generated by functions over the latent factors or embeddings of users and items. Recent advancements in deep learning, especially *Graph Convolutional Networks* (GCN) [Defferrard *et al.*, 2016; Bronstein *et al.*, 2017; Kipf and Welling, 2017; Hamilton *et al.*, 2017], have brought new ideas for tackling this essential artificial intelligence problem. GCN generalizes the definition of convolution from the regular grid to irregular grid, like graph structures. The GCN framework generates node representations by a localized parameter-sharing operator, known as *g*raph aggregator [Hamilton *et al.*, 2017; Zhang *et al.*, 2018]. A graph aggregator calculates a node's representation by transforming and aggregating the features of its local neighborhoods. By stacking multiple graph aggregators and nonlinear functions, we build a deep neural network that can extract features across far reaches of a graph. Because the local neighborhood set can be viewed as the receptive field of a convolution kernel, this kind of neighborhood aggregation methods is named as *graph convolution*, which also have connections to spectral graph theory [Kipf and Welling, 2017].

Monti et al. [2017] proposed the first GCN-based method for recommender systems. In their approach, GCN was used to aggregate information from two auxiliary user-user and item-item graphs. The latent factors of users and items were updated after each aggregation step, and a combined objective function of GCN and MF was used to train the model. After that, Berg et al. [2017] proposed the *Graph Convolutional Matrix Completion* (GC-MC) model. GC-MC directly characterized the relationship between users and items as a bipartite interaction graph. Two multi-link graph convolution layers were used to aggregate user features and item features. The ratings were estimated by predicting the edge labels. Thanks to the power of GCN in learning high-quality user and item representations, GC-MC has achieved state-of-the-art performance in several public recommendation benchmarks.

While being powerful, the GC-MC model has two significant limitations. To distinguish each node, the model uses one-hot vectors as node input. This makes the input dimensionality proportional to the total number of nodes and thus is not scalable to large graphs. Moreover, the model is unable to predict the ratings for new users or items that are not seen in the training phase because we cannot represent unknown nodes as one-hot vectors. The task of predicting ratings for new users or items is also known as the *cold start* problem.

In this paper, we propose a new architecture, *STAcked and Reconstructed Graph Convolutional Networks* (STAR-GCN), to solve these problems. Unlike GC-MC, STAR-GCN directly learns low-dimensional user and item embeddings as the input to the network in an end-to-end fashion. To improve the learned embeddings and also generalize the model to predict

embeddings of unseen nodes for the cold start problem, STAR-GCN masks a part of or the whole user and item embeddings and reconstructs these masked embeddings with a block of graph encoder-decoder in the training phase. This technique is inspired by the recent success of the 'masked language model' in learning language embeddings [Devlin *et al.*, 2018]. Moreover, we build a stack of encoder-decoder blocks in conjunction with intermediate task-specific supervision to enhance the final performance. During implementation, we find that training the GCN-based models for rating prediction faces the label leakage issue, which results in the overfitting problem and significantly degrades the final performance. To avoid the leakage issue, we provide a *sample-and-remove* training strategy and empirically demonstrate the effectiveness.

We conduct experiments over two tasks: transductive rating prediction and inductive rating prediction. The transductive rating prediction is generally used in traditional matrix completion tasks, i.e., all the testing users and items are observed in training data. The inductive rating prediction is a newly introduced task to evaluate different models' ability on the cold start problem. We ask new users to rate a few items or require new items to be rated by a few users. These data are only used in the inference step to elicit initial information about new users/items, which is similar to the ask-to-rate technique [Nadimi-Shahraki and Bahadorpour, 2014] for cold start. Experiments show that STAR-GCN achieves state-of-the-art performance in four out of five real-world datasets in the transductive setting. In the inductive setting, our STAR-GCN consistently and significantly outperforms the baselines.

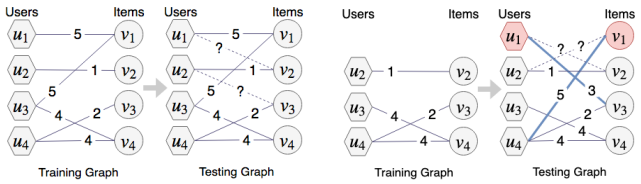
Our main contributions are: (1) we propose a new architecture for recommender systems to learn latent factors of users and items in both transductive and inductive settings; (2) we are the first to explicitly pinpoint a training label leakage issue when implementing GCN-based models in rating prediction tasks and propose a training strategy to avoid this issue, leading to substantial performance improvement; (3) our STAR-GCN models achieve state-of-the-art performance in four out of five real-world recommendation datasets in the transductive setting and significantly outperform other models in the inductive setting.

2 Preliminary

We denote vectors with bold lowercase letters, matrices with bold uppercase letters, and sets with calligraphy letters. We omit the bias variable of linear transformation for brevity.

2.1 Rating Prediction Tasks

GCN-based models treat the recommendation environment as an undirected bipartite graph G that contains two disjoint node sets, users $\mathcal{U}$ and items $\mathcal{V}$. Suppose there are N users and M items, an edge value $r_{i,j} \in \mathcal{R}$ represents an observed rating value from user u_i to item v_j. The rating set $\mathcal{R}$ may contain several ordinal rating levels, i.e., $r \in \{1, ..., R\} = \mathcal{R}$. Each rating level indicates a link type in the bipartite graph. All the training rating pairs form a training graph, and they are included in a testing graph. Examples are shown in Figure 1. The goal of rating prediction is to predict the ratings a user would give to other items, given a small subset of observed



(a) Transductive rating prediction (b) Inductive rating prediction

Figure 1: Two rating prediction tasks. (a) All users and items are observed in the training graph. (b) u_1 and v_1 are not seen in the training time. But we can access a few edges connected with the new nodes in the testing graph before we make predictions.

rating pairs. We focus on two types of rating prediction tasks: transductive rating prediction and inductive rating prediction. Figure 1 illustrates the difference between these two tasks.

Transductive Rating Prediction. Figure 1a shows a transductive rating prediction example, where users and items appearing in the testing graph are observed in the training graph. Prior collaborative filtering methods, like *Matrix Factorization* (MF) [Koren *et al.*, 2009], primarily concentrate on this task.

Inductive Rating Prediction. Figure 1b illustrates an example of the inductive rating prediction task. u_1 and v_1 are two new nodes, which are not seen at the training time but appear in the testing rating pairs. Before making predictions, we access a few rated edges connected with these new nodes in the testing graph. Traditional collaborative filtering methods cannot solve this task without re-training the models. The standard way is to rely on content information to model the users' and items' preferences. The *Collaborative Deep Learning* (CDL) [Wang *et al.*, 2015] model and *DropoutNet* [Volkovs *et al.*, 2017] are two recent representative methods. The core idea of these two models is to use a deep neural network to learn effective features of the input node content.

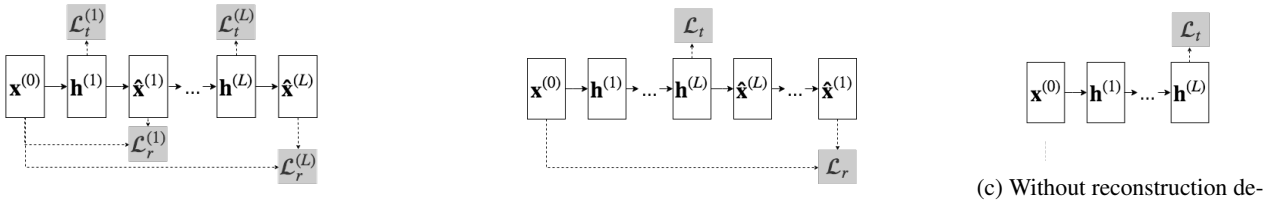
Recent progress in deep learning on graphs, mainly the GCN models, can address the above two tasks by learning transductive and inductive node representations. STAR-GCN inherits the ability of GCN for both transductive and inductive learning. Compared with CDL and DropoutNet, our STAR-GCN not only takes account of the node's content information but also utilizes the structural information to learn the embeddings of new nodes. Thus, STAR-GCN can solve the cold start problem when the content information is unavailable, which is infeasible for CDL and DropoutNet.

2.2 Graph Convolutional Matrix Completion

In this subsection, we briefly revisit GC-MC [Berg *et al.*, 2017]. Our STAR-GCN employs a similar graph aggregator to encode structural information. GC-MC uses a multi-link graph convolutional encoder to produce node representations. Each link type r is assigned with a specific transformation. The messages from items to user u_i are computed as

$$\begin{aligned} \mathbf{y}_{u_i}^r &= \sum_{v_j \in \mathcal{N}_r(u_i)} \frac{1}{c_{ij}} \mathbf{W}_a^r \mathbf{x}_{v_j}, \\ \mathbf{h}_{u_i} &= \alpha(\mathbf{W}_h \alpha(\sum_{r=1}^{R} \mathbf{y}_{u_i}^r)). \end{aligned} \tag{1}$$

Here, $\mathbf{y}_{u_i}^r$ is the aggregated output of a link type r. $\mathbf{x}_{v_j} \in \mathbb{R}^{d_{in}}$ is the initial vector of v_j and d_{in} is the input dimension. $\mathbf{W}_a^r$



(a) Our STAR-GCN architecture with L blocks (b) One block with L-layer encoder and decoder (c) Without reconstruction decoder

Figure 2: Model Architectures. $\mathbf{x}^{(0)}$ is the initial node vector. $\mathbf{h}^{(\ell)}$ is the output of the ℓ-th graph encoder and $\hat{\mathbf{x}}^{(\ell)}$ is the output node vector of the ℓ-th decoder. $\mathcal{L}_t$ is a task specific loss and $\mathcal{L}_r$ is a reconstruction loss. (b) and (c) are two special cases of (a).

of size $d_a \times d_{in}$ is a link-specific weight matrix for rating level r, which transforms a vector of the dimension d_{in} to an hidden size d_a. c_{ij} is a normalized constant, computed as $\sqrt{|\mathcal{N}_r(u_i)||\mathcal{N}_r(v_j)|}$ with $\mathcal{N}_r(\cdot)$ denoting a set of neighbors connected by edge-value r. After computing the link specific messages, we sum the messages from total R types of links and pass the output to a non-linear function $\alpha(\cdot)$. Finally, we employ a fully connected layer with parameter $\mathbf{W}_h$ of size $d_h \times d_a$ and another non-linear activation to produce the final node vector $\mathbf{h}_{u_i}$ for user u_i. Messages from users to items are processed analogously with a separate set of parameters.

In the next step, the GC-MC model takes the computed user vector $\mathbf{h}_{u_i}$ and item vector $\mathbf{h}_{v_j}$ as the input to predict the rating value $\hat{r}_{i,j}$. See Berg et al. [2017] for more details. In the following section, we do not distinguish user u and item v and call them as a node.

3 Our Models

The architecture of STAR-GCN is a multi-block graph encoder-decoder shown in Figure 2a. The multi-block architecture allows reassessment of initial estimates and features across the whole graph. In particular, each block contains two components: a graph encoder and a decoder. The graph encoder generates node representations by encoding semantic graph structures as well as input content features, and the decoder aims to recover the input node embeddings. For each block, we impose a task-specific loss after graph encoders and a node reconstruction loss after decoders.

STAR-GCN supports two different types of combinations between two consecutive blocks, by stacking or by recurrence. The main difference is whether to share parameters among blocks or not. By stacking, we consecutively place multiple encoder-decoder blocks with separate sets of parameters. By recurrence, we unfold a single encoder-decoder block, so the same set of parameters are shared across all the blocks, which curtails the total memory usage. Besides, our STAR-GCN is a general framework, which can be simplified to some individual cases, as shown in Figure 2b and 2c. Empirical studies on two combinations and the simplified models are in Table 3.

3.1 Input Node Representations

To make the network scalable to large graphs, we use an embedding lookup table $\mathbf{X}_e \in \mathbb{R}^{d_e \times |M+N|}$ to map each node to a low-dimensional vector $\mathbf{x}_e \in \mathbb{R}^{d_e}$, where $d_e \ll |M+N|$. $\mathbf{X}_e$ is trained end-to-end along with the network. However, naively replacing the one-hot vectors with embeddings, fails to tackle the cold start problem because we cannot set the embeddings of nodes that are not seen in the training phase.

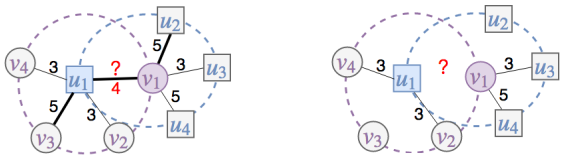
So, to generalize the embedding learning technique to new nodes and preserve the high prediction accuracy, we take an approach of masking some percentage of the input nodes at random and then reconstructing the clean node embeddings. Like Devlin et al. [2018], in each training batch, we mask P_m percentage, say 20%, of the whole input nodes at random. Then we reconstruct these masked embeddings. For the masked nodes, they perform the following choices: (1) with probability p_z, we set the node embeddings to be zero; and (2) with the remaining probability, we keep the node unchanged.

Training with the masked embedding mechanism has two advantages. First, it can learn embeddings for nodes that are not observed in the training phase. In a cold start scenario, we initialize the embeddings of new nodes to be zero and gradually refine the estimated embeddings by multiple blocks of GCN encoder-decoders. For instance, the first block predicts the embedding of the new node by leveraging the neighborhood data (or node attributes, if available). Then the predicted embedding is fed to the second block to predict ratings and a refined embedding. The rating and embedding prediction losses are jointly optimized. Thus, STAR-GCN can solve the cold-start issue by iteratively refining the embeddings and is fundamentally different from GC-MC. Second, STAR-GCN leads to improvement in the transductive setting. In the training phase, part of the node embeddings are masked and the network is asked to reconstruct these masked embeddings, which requires the network to encode the relationships between users and items effectively. Thus, the reconstruction loss acts as a multi-task regularizer that improves the performance of the primary rating prediction task.

When external node features are available, they are first processed via a separate network and then concatenated with the node embeddings. The feature vector $\mathbf{f}$ is mapped to a fixed size vector $\mathbf{x}_f \in \mathbb{R}^{d_f}$ using a two-layer feedforward neural network, i.e., $\mathbf{x}_f = \mathbf{W}_2\alpha(\mathbf{W}_1\mathbf{f})$, where both layers have an output dimension d_f. Now the input node vector $\mathbf{x}$ becomes $[\mathbf{x}_e; \mathbf{x}_f]$ and the input dimension $d_{in} = d_e + d_f$ rather than $\mathbf{x}_e$ when the content information is not available.

3.2 Graph Encoder and Decoder

The graph encoder transforms an input node vector $\mathbf{x} \in \mathbb{R}^{d_{in}}$ into hidden state $\mathbf{h}$ of size d_h by aggregating neighboring information of different rating levels, i.e., $\mathbf{h} = \text{Enc}(\mathbf{x})$. We choose the encoder to be the multi-link GCN aggregator in GC-MC, which is formulated in Eq.(1). A decoder maps the structural-encoded node representation $\mathbf{h}$ to a d_{in}-dimensional



(a) The original node and edge input to an aggregator, including the ground truth label 4.

(b) The nodes and edges to be aggregated after removing the sampled edges.

Figure 3: The graph aggregation operator for u_1 and v_1 when predicting the rating value between u_1 and v_1. The dash circles include the nodes and edges to be aggregated. The links in bold lines are the sampled edges (rating pairs) to be trained. (a) illustrates the leakage issue. (b) gives a solution by removing the sampled edges.

reconstructed embedding vector $\hat{\mathbf{x}}$, i.e., $\hat{\mathbf{x}} = \text{Dec}(\mathbf{h})$. We use a two-layer feedforward neural network as a decoder, i.e., $\hat{\mathbf{x}} = \mathbf{W}_4\alpha(\mathbf{W}_3\mathbf{h})$, where the output dimensions are both d_{in}.

STAR-GCN is a general framework with a stack of GCN encoder-decoder blocks. Any variant of GCNs, e.g., GraphSAGE [Hamilton *et al.*, 2017] and GAT [Veličković *et al.*, 2018], can be used as an encoder or decoder in STAR-GCN. Our graph encoder-decoder is different from the graph auto-encoder model [Kipf and Welling, 2016] mainly in the role of the decoder. Our decoder is to recover the initial input node vectors, while their decoder is a task-specific classifier to produce predictions. Another difference is that our graph aggregator considers different link types [Schlichtkrull *et al.*, 2018], whereas their aggregator only models single link type.

3.3 Loss

Suppose there are L blocks, the loss function is expressed as

$$\mathcal{L} = \sum_{\ell=1}^{L}(\mathcal{L}_t^{(\ell)} + \lambda^{(\ell)}\mathcal{L}_r^{(\ell)}), \tag{2}$$

where $\mathcal{L}_t^{(\ell)}$ is a supervised task-specific loss, i,e., the rating prediction loss, and $\mathcal{L}_r^{(\ell)}$ is a reconstruction loss from the ℓ-th block. $\lambda^{(\ell)}$ is a constant weighting factor. In the following description, we omit the layer superscript for brevity.

Suppose we have a batch of sampled edges $\mathcal{E}_b$ and two sets of masked nodes $\mathcal{U}_m$ and $\mathcal{V}_m$, the specific losses are

$$\mathcal{L}_t = \frac{1}{|\mathcal{E}_b|}\sum_{(u_i,v_j)\in\mathcal{E}_b}(r_{i,j} - \mathbf{u}_i^T\mathbf{v}_j)^2,$$

$$\mathcal{L}_r = \frac{1}{2|\mathcal{U}_m|}\sum_{u\in\mathcal{U}_m}||\mathbf{x}_u - \hat{\mathbf{x}}_u||^2 + \frac{1}{2|\mathcal{V}_m|}\sum_{v\in\mathcal{V}_m}||\mathbf{x}_v - \hat{\mathbf{x}}_v||^2.$$

For the rating prediction loss, $\mathbf{u}_i$ and $\mathbf{v}_j$ are generated by linear transforms with the output of a graph encoder, i.e, $\mathbf{u}_i = \mathbf{W}_r^u\mathbf{h}_{u_i}$ and $\mathbf{v}_j = \mathbf{W}_r^v\mathbf{h}_{v_j}$ with user or item-specific matrix parameters $\mathbf{W}_r^u$, $\mathbf{W}_r^v \in \mathbb{R}^{d_r\times d_o}$. We train the overall models end-to-end with backpropagation.

During the inference period, a L-block STAR-GCN can produce L predictions. Generally, we take the prediction from the last block as a final result.

3.4 Training by Avoiding a Leakage Issue

When training the GCN-based models for rating prediction, we discover a training label leakage issue. The ground-truth training labels are involved in the model input, which significantly degrades the testing prediction performance. Specifically, suppose we have a training input x and a label y, the model should be trained as $y = f_\theta(x)$. However, if the training label leakage occurs, the model becomes $y = f_\theta(x, y)$ resulting in an overfitting problem. The label leakage problem is a GCN specific issue because of the neighborhood aggregation operator. The user-item rating values in the training set are used to construct edges of the bipartite graph. Thus, when we utilize the neighboring data to update a node's representation, the rating values, which we need to predict, are included in the graph structure. This causes the label leakage issue. As in Figure 3a, y is the edge value 4 between u_1 and v_1, which is taken as the input to a graph aggregator.

To avoid the leakage issue, we provide a *sample-and-remove* training strategy. At each iteration, we sample a fixed size batch of rating pairs and remove the sampled pairs (edges) from the training graph before we start training the model. As in Figure 3b, the sampled edges with bold links are removed from the graph when we aggregate neighbors. After avoiding this leakage issue, the network shows a substantial boost in performance. The compared results of the leakage issue are given in Table 3.

Table 1: Statistics of the datasets. 'D_U' and 'D_V' are the input feature dimension of users and items, respectively.

	D_U	D_V	#U	#V	$\mathcal{R}$	#R
Flixster	3K	3K	2,341	2,956	0.5,1,..,5	26,173
Douban	3K	-	2,999	3,000	1,..,5	136,891
ML-100K	23	320	943	1,682	1,..,5	100K
ML-1M	23	320	6,040	3,706	1,..,5	1M
ML-10M	-	321	69,878	10,677	0.5,1,..,5	10M

4 Experiments

We conduct extensive experiments on five popular recommendation benchmarks for the transductive and inductive rating prediction tasks. The datasets are summarized in Table 1. **Flixster** and **Douban** are preprocessed and provided by Monti et al. [2017]. The user and item features are the adjacency vectors of the respective user-user and item-item graphs. The MovieLens[1] datasets contain different scales of rating pairs, i.e., 100 thousand, 1 million, and 10 million. We denote them as **ML-100K**, **ML-1M**, and **ML-10M**. For user features, we take the *age* as a scalar and the *gender* as a binary numerical value, and the *occupation* as a one-hot encoding vector. For movie features, we concatenate the title name, release year, and one-hot encoded genres. We process title names by averaging the off-the-shelf 300-dimensional GloVe CommonCrawl word vector [Pennington *et al.*, 2014] of each word.

We take the commonly adopted *Root Mean Squared Error* (RMSE) metric to evaluate the prediction accuracy between the ground truth r_i and the predicted rating $\hat{r}_i$,

$$\text{RMSE} = \sqrt{\frac{\sum_{i=1}^{N}(\hat{r}_i - r_i)^2}{N}}. \tag{3}$$

[1]Movielens [Harper and Konstan, 2016]: https://grouplens.org/datasets/movielens/

Table 2: Comparison of test RMSE scores for transductive rating prediction.

	Flixster	Douban	ML-100K	ML-1M	ML-10M
BiasMF [Koren *et al.*, 2009]	-	-	0.917	0.845	0.803
NNMF [Dziugaite and Roy, 2015]	-	-	0.907	0.843	-
I-AUTOREC [Sedhain *et al.*, 2015]	-	-	-	0.831	0.782
GRALS [Rao *et al.*, 2015]	1.245	0.833	0.945	-	-
CF-NADE [Zheng *et al.*, 2016]	-	-	-	**0.829**	0.771
Factorized EAE [Hartford *et al.*, 2018]	-	-	0.910	0.860	-
sRMGCNN [Monti *et al.*, 2017]	0.926	0.801	0.929	-	-
GC-MC [Berg *et al.*, 2017]	0.917	0.734	0.910	0.832	0.777
STAR-GCN	**0.879**±0.0030	**0.727**±0.0006	**0.895**±0.0009	0.832±0.0016	**0.770**±0.0001

Table 3: Ablation analysis of different STAR-GCN models in transductive rating prediction. 'b' denotes the *block* number, 'l' denotes the number of graph convolution *layer*s in each block, and '-' denotes minus. '- rec.' means the model does not have the reconstruction module and '- rm.' denotes the model is trained without removing sampled edges. 'recurrent' denotes that the blocks are combined by recurrence. Otherwise, the blocks are combined by stacking.

	Models	Flix.	Dou.	ML-100K	ML-1M	ML-10M
Emb. only	1b2l (- rec., - rm.)	0.920	0.731	0.921	0.841	0.782
	1b2l (- rec.)	0.893	0.728	0.899	0.835	0.778
	1b2l	0.891	0.728	0.901	0.834	0.771
	2b1l (recurrent)	0.883	**0.727**	**0.895**	0.833	0.773
	2b1l	**0.879**	**0.727**	0.898	**0.832**	0.771
With Fea.	1b2l (- rec.,- rm.)	0.917	0.731	0.920	0.840	0.782
	1b2l (- rec.)	0.889	0.727	0.899	0.835	0.778
	1b2l	0.887	0.728	0.901	0.834	**0.770**
	2b1l (recurrent)	**0.879**	0.728	0.896	0.833	0.772
	2b1l	0.880	**0.727**	0.896	**0.832**	0.771

4.1 Model Architecture and Implementation Details

We test the overall network design with different sets of hyperparameters. The validation set determines our final design. We regard Flixtser, Douban, and ML-100K as small datasets and ML-1M and ML-10M as large datasets. After tuning the hyperparameters, we roughly apply two sets of hyperparameters, one for small datasets and the other for large datasets. In all models, we choose the non-linear function $\alpha(\cdot)$ as a LeakyReLU activation with the negative slope equals to 0.1. For the input vectors, we set the dimension of node embeddings d_e to be 32 for small datasets and 64 for large datasets. When incorporating features, we take the projection dimension d_f to be 8 for small datasets and 32 for large datasets. Regarding the masking mechanism, for transductive prediction, we randomly select P_m=10% of the nodes for reconstruction and set p_z=0. For inductive prediction, we uniformly choose P_m=40% nodes and mask them to be zero, i.e., p_z=1, to approximate the testing data distribution. For encoders, the hidden size d_a sets to be 250. The dimension of the output layer d_h sets to be 75. We apply a dropout layer to the input of a GCN layer with a dropout rate of 0.5 for small datasets and 0.3 for large datasets. For decoders, all the hidden sizes are fixed as the node input dimension, i.e., d_{in}. When predicting ratings, we set the projection size d_r to be 64.

We train the STAR-GCN models with Adam [Kingma and Ba, 2015] optimizer and use the validation set to perform learning rate decay scheduler. The initial learning rate is set to be 0.002 and gradually decreases to be 0.0005 with the decay rate of 0.5 each time the validation RMSE score does not fall in a window of 100 iterations, and the early stopping occurs for 150 iterations. The gradient normalization value clips to be no larger than 1.0. The training batch size is fixed to be 10K for small datasets, 100K for ML-1M, and 500K for ML-10M. We train the model three times with different random seeds, except for ML-10M training two times, and report the average test RMSE scores along with the standard deviation.

4.2 Transductive Rating Prediction

All the datasets are used for transductive rating prediction evaluations, where the users and items are all observed in the training dataset. For fair comparison, we strictly follow the experimental setup of Berg et al. [2017]. For Douban and Flixster, we use the split test set provided by Monti et al. [2017] with 10% of rating pairs as the testing set. For ML-100K, we use the first of the five provided data split with 20% for testing. For ML-1M and ML-10M, we randomly split the edges with 10% for testing. In the transductive setting, we perform a thorough comparison of STAR-GCN with multiple baselines and state-of-the-art models listed in Table 2. Baseline Results are taken from Berg et al. [2017]. We also conduct comprehensive ablation analysis in Table 3.

Table 2 summarizes all results of the baselines and our STAR-GCN. The baseline scores are directly taken from Monti et al. [2017] and Berg et al. [2017]. The results of STAR-GCN are produced by different variant models with a total of two graph convolutional layers, labeled as either '1b2l' or '2b1l' in Table 3. '1b2l' denotes a model with one block of encoder-decoder and each encoder includes a two-layer GCN, as in Figure 2b. In contrast, '2b1l' indicates a model with two encoder-decoder blocks and each encoder contains a one-layer GCN, as in Figure 2a. In particular, '1b2l (- rec.)' indicates the model only has an encoder without the reconstruction module, as in Figure 2c. The reported RMSE scores for STAR-GCN in Table 2 is the best results of different STAR-GCN models listed in Table 3. We note that the proposed STAR-GCN architecture achieves the state-of-the-art results on four out of five datasets. We have the following findings from Table 3.

Effect of removing sampled training edges within mini-batches. Comparing the results of '1b2l (- rec. - rm.)' and '1b2l (- rec.)', we see a significant decrease in the testing RMSE scores after removing the sampled user-item pairs from the bipartite graph in each training batch. This proves the

Table 4: Comparison of test RMSE scores for inductive rating prediction. '- rec.' denotes the model does not have the reconstruction module and '+ fea.' means the model uses external node features. We train all models three times and report the mean scores and the standard deviation.

Datasets	Models	Items 20%			Users 20%		
		50%	30%	10%	50%	30%	10%
Douban	DropoutNet	-	-	-	0.797±0.002	0.797±0.003	0.797±0.001
	CDL	-	-	-	0.781±0.006	0.781±0.001	0.781±0.001
	STAR-GCN(- rec.)	0.734±0.001	0.746±0.001	0.777±0.002	0.731±0.000	0.738±0.000	0.753±0.001
	STAR-GCN(- rec., + fea.)	-	-	-	0.731±0.002	0.737±0.000	0.753±0.001
	STAR-GCN	**0.725**±0.001	**0.734**±0.001	**0.764**±0.000	**0.725**±0.001	**0.731**±0.001	0.747±0.001
	STAR-GCN(+ fea.)	-	-	-	**0.725**±0.002	**0.731**±0.000	**0.746**±0.000
ML-100K	DropoutNet	1.223±0.065	1.167±0.031	1.144±0.024	1.015±0.002	1.022±0.006	1.023±0.003
	CDL	1.083±0.009	1.082±0.007	1.082±0.007	1.011±0.005	1.013±0.006	1.015±0.004
	STAR-GCN(- rec.)	0.932±0.001	0.943±0.001	0.976±0.003	0.919±0.002	0.933±0.001	0.949±0.001
	STAR-GCN(- rec., + fea.)	0.928±0.002	0.941±0.002	0.977±0.004	0.916±0.005	0.931±0.004	0.951±0.005
	STAR-GCN	0.919±0.001	**0.926**±0.000	**0.954**±0.001	**0.907**±0.004	**0.917**±0.005	0.937±0.005
	STAR-GCN(+ fea.)	**0.918**±0.002	**0.926**±0.002	0.956±0.000	**0.907**±0.002	**0.917**±0.001	**0.936**±0.004
ML-1M	DropoutNet	1.169±0.120	1.134±0.034	1.256±0.128	1.002±0.001	1.005±0.005	1.003±0.001
	CDL	1.068±0.009	1.069±0.009	1.068±0.009	0.974±0.000	0.975±0.000	0.974±0.000
	STAR-GCN(- rec.)	0.862±0.001	0.872±0.004	0.903±0.004	0.859±0.002	0.868±0.001	0.891±0.001
	STAR-GCN(- rec., + fea.)	0.861±0.002	0.867±0.002	0.910±0.006	0.859±0.001	0.869±0.001	0.893±0.001
	STAR-GCN	**0.844**±0.000	**0.850**±0.000	**0.876**±0.004	**0.848**±0.001	**0.858**±0.001	**0.882**±0.000
	STAR-GCN(+ fea.)	**0.844**±0.001	0.851±0.001	**0.876**±0.002	0.849±0.001	**0.858**±0.000	0.883±0.001

effectiveness of our sample-remove training strategy to avoid the training data leakage issue.

Effect of reconstructing masked nodes. Comparison of the results of the models labeled '- rec.' with the models having no such label indicates that the models possessing the reconstruction module consistently beat the models without reconstruction utilizing the same total number of graph encoders, which proves that our reconstruction mechanism is beneficial to the final prediction performance.

Effect of the recurrent structure. Comparing the results of '2b1l (recurrent)' and '2b1l', we see that the recurrent structure can achieve competitive results with fewer parameters.

Effect of incorporating features. By comparing the results from the same models with and without features, we note that combining external node features does not always produce better performance.

4.3 Inductive Rating Prediction

We conduct inductive rating prediction experiments on three datasets, Douban, ML-100K, and ML-1M. For each dataset, we keep 20% of user (or item) nodes as the testing nodes and remove them from the training graph. Then, we choose a fraction of ratings linked with the testing nodes as the edges that are observed in the testing phase. Ratings that are not chosen by this step are kept as the testing data. The predictor never sees these 20% nodes and needs to rely on the observed links (together with the node features, if available) to predict ratings. We conduct experiments on three different fractions, 50%, 30%, and 10%, which means that there are 50%, 30%, and 10% ratings linked with new nodes available to the predictor in the testing phase. Intuitively, the more edges we access, the more information we have, and the better the performance will be. We implement two baseline models, CDL [Wang *et al.*, 2015] and DropoutNet [Volkovs *et al.*, 2017], and compare some variants of our STAR-GCN architecture. We use the '2b1l' versions of STAR-GCN in this task.

The testing RMSE scores are listed in Table 4. We find a worse performance tendency when the predictor accesses fewer neighboring edges for the new users/items in the testing phase. We show that our STAR-GCN model produces significantly better results than two baselines. Moreover, comparing the results of the models with and without reconstruction modules, we find that the reconstruction mechanism plays a crucial role in improving the final performance. An interesting observation is that incorporating content information for new nodes is not always beneficial to the final results. The reason may be that our reconstructed node embeddings have already contained enough information for accurate predictions, which proves that our STAR-GCN can effectively solve the cold start problem using structural information.

5 Conclusion and Future Work

We introduce a new GCN-based architecture and apply it to transductive and inductive rating prediction. Our STAR-GCN achieves the state-of-the-art results in both tasks. Our architecture is generic and can be used in other applications, such as abnormal behavior detection, spatiotemporal forecasting [Shi and Yeung, 2018], thread popularity prediction [Chan and King, 2018], and so on [Gao *et al.*, 2018]. Moreover, we discover a training label leakage issue when implementing GCN-based models for rating prediction tasks. The discovery should serve as a reminder of later research. In the future, we plan to improve our STAR-GCN to handle heterogeneous graphs with diverse node types for better simulating real-life scenarios and to integrate ranking algorithms [Su *et al.*, 2017] to solve other recommendation tasks.

6 Acknowledgement

The work described in this paper was partially supported by the Research Grants Council of the Hong Kong Special Administrative Region, China (No. CUHK 14208815 of the General Research Fund) and Meitu (No. 7010445).

References

[Berg *et al.*, 2017] Rianne van den Berg, Thomas N Kipf, and Max Welling. Graph convolutional matrix completion. *arXiv preprint arXiv:1706.02263*, 2017.

[Bronstein *et al.*, 2017] Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017.

[Candès and Recht, 2009] Emmanuel J Candès and Benjamin Recht. Exact matrix completion via convex optimization. *Foundations of Computational mathematics*, 9(6):717, 2009.

[Chan and King, 2018] Hou Pong Chan and Irwin King. Thread popularity prediction and tracking with a permutation-invariant model. In *EMNLP*, pages 3392–3401, 2018.

[Defferrard *et al.*, 2016] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *NIPS*, pages 3844–3852, 2016.

[Devlin *et al.*, 2018] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[Dziugaite and Roy, 2015] Gintare Karolina Dziugaite and Daniel M Roy. Neural network matrix factorization. *arXiv preprint arXiv:1511.06443*, 2015.

[Gao *et al.*, 2018] Yifan Gao, Jianan Wang, Lidong Bing, Irwin King, and Michael R Lyu. Difficulty controllable question generation for reading comprehension. *arXiv preprint arXiv:1807.03586*, 2018.

[Hamilton *et al.*, 2017] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *NIPS*, pages 1025–1035, 2017.

[Harper and Konstan, 2016] F Maxwell Harper and Joseph A Konstan. The movielens datasets: History and context. *Acm transactions on interactive intelligent systems (tiis)*, 5(4):19, 2016.

[Hartford *et al.*, 2018] Jason S. Hartford, Devon R. Graham, Kevin Leyton-Brown, and Siamak Ravanbakhsh. Deep models of interactions across sets. In *ICML*, pages 1914–1923, 2018.

[Kingma and Ba, 2015] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

[Kipf and Welling, 2016] Thomas N Kipf and Max Welling. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308*, 2016.

[Kipf and Welling, 2017] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017.

[Koren *et al.*, 2009] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 8(Sep):30–37, 2009.

[Monti *et al.*, 2017] Federico Monti, Michael Bronstein, and Xavier Bresson. Geometric matrix completion with recurrent multi-graph neural networks. In *NIPS*, pages 3697–3707, 2017.

[Nadimi-Shahraki and Bahadorpour, 2014] Mohammad-Hossein Nadimi-Shahraki and Mozhde Bahadorpour. Cold-start problem in collaborative recommender systems: efficient methods based on ask-to-rate technique. *Journal of computing and information technology*, 22(2):105–113, 2014.

[Pennington *et al.*, 2014] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *EMNLP*, pages 1532–1543, 2014.

[Rao *et al.*, 2015] Nikhil Rao, Hsiang-Fu Yu, Pradeep K Ravikumar, and Inderjit S Dhillon. Collaborative filtering with graph information: Consistency and scalable methods. In *NIPS*, pages 2107–2115, 2015.

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

[Sedhain *et al.*, 2015] Suvash Sedhain, Aditya Krishna Menon, Scott Sanner, and Lexing Xie. Autorec: Autoencoders meet collaborative filtering. In *WWW*, pages 111–112, 2015.

[Shi and Yeung, 2018] Xingjian Shi and Dit-Yan Yeung. Machine learning for spatiotemporal sequence forecasting: A survey. *arXiv preprint arXiv:1808.06865*, 2018.

[Su *et al.*, 2017] Yuxin Su, Irwin King, and Michael R. Lyu. Learning to rank using localized geometric mean metrics. In *SIGIR*, pages 45–54, 2017.

[Veličković *et al.*, 2018] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. In *ICLR*, 2018.

[Volkovs *et al.*, 2017] Maksims Volkovs, Guangwei Yu, and Tomi Poutanen. Dropoutnet: Addressing cold start in recommender systems. In *NIPS*, pages 4957–4966, 2017.

[Wang *et al.*, 2015] Hao Wang, Naiyan Wang, and Dit-Yan Yeung. Collaborative deep learning for recommender systems. In *KDD*, pages 1235–1244. ACM, 2015.

[Zhang *et al.*, 2018] Jiani Zhang, Xingjian Shi, Junyuan Xie, Hao Ma, Irwin King, and Dit-Yan Yeung. Gaan: Gated attention networks for learning on large and spatiotemporal graphs. In *UAI*, pages 339–349, 2018.

[Zheng *et al.*, 2016] Yin Zheng, Bangsheng Tang, Wenkui Ding, and Hanning Zhou. A neural autoregressive approach to collaborative filtering. In *ICML*, pages 764–773, 2016.