Pre-Training Graph Neural Networks for Cold-Start Users and Items Representation

Bowen Hao
Renmin University of China
jeremyhao@ruc.edu.cn

Jing Zhang*
Renmin University of China
zhang-jing@ruc.edu.cn

Hongzhi Yin
The University of Queensland
h.yin1@uq.edu.au

Cuiping Li
Renmin University of China
licuiping@ruc.edu.cn

Hong Chen
Renmin University of China
chong@ruc.edu.cn

ABSTRACT

Cold-start problem is a fundamental challenge for recommendation tasks. Despite the recent advances on Graph Neural Networks (GNNs) incorporate the high-order collaborative signal to alleviate the problem, the embeddings of the cold-start users and items aren't explicitly optimized, and the cold-start neighbors are not dealt with during the graph convolution in GNNs. This paper proposes to pre-train a GNN model before applying it for recommendation. Unlike the goal of recommendation, the pre-training GNN simulates the cold-start scenarios from the users/items with sufficient interactions and takes the embedding reconstruction as the pretext task, such that it can directly improve the embedding quality and can be easily adapted to the new cold-start users/items. To further reduce the impact from the cold-start neighbors, we incorporate a self-attention-based meta aggregator to enhance the aggregation ability of each graph convolution step, and an adaptive neighbor sampler to select the effective neighbors according to the feedbacks from the pre-training GNN model. Experiments on three public recommendation datasets show the superiority of our pre-training GNN model against the original GNN models on user/item embedding inference and the recommendation task.

CCS CONCEPTS

• **Information systems** → **Social recommendation**;

KEYWORDS

Pre-training, graph neural networks, cold-start, recommendation
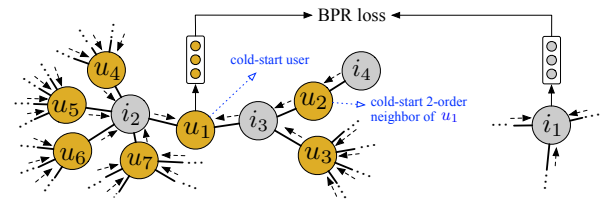
---

*Corresponding Author.

Figure 1: A GNN model for recommendation.

1 INTRODUCTION

Recommendation systems [14, 21] have been extensively deployed to alleviate information overload in various web services, such as social media, E-commerce websites and news portals. To predict the likelihood of a user adopting an item, collaborative filtering (CF) is the most widely adopted principle. The most common paradigm for CF, such as matrix factorization [21] and neural collaborative filtering [14], is to learn embeddings, i.e. the preferences for users and items and then perform the prediction based on the embeddings [13]. However, these models fail to learn high-quality embeddings for the cold-start users/items with sparse interactions.

To address the cold-start problem, traditional recommender systems incorporate the side information such as content features of users and items [40, 44] or external knowledge graphs (KGs) [35, 37] to compensate the low-quality embeddings caused by sparse interactions. However, the content features are not always available, and it is not easy to link the items to the entities in KGs due to the incompleteness and ambiguation of the entities.

On another line, inspired by the recent development of graph neural networks (GNNs) [2, 11, 19], NGCF [38] and LightGCN [13] encode the high-order collaborative signal in the user-item interaction graph by a GNN model, based on which they perform the recommendation task. As shown in Fig. 1, a typical recommendation-oriented GNN conducts graph convolution on the local neighborhood's embeddings of u_1 and i_1. Through iteratively repeating the convolution by multiple steps, the embeddings of the high-order neighbors are propagated to u_1 and i_1. Based on the aggregated embeddings of u_1 and i_1, the likelihood of u_1 adopting i_1 is estimated, and cross-entropy loss [3] or BPR loss [13, 38] is usually adopted to compare the likelihood and the true observations.

Despite the success of capturing the high-order collaborative signal in GNNs [13, 38], the cold-start problem is not thoroughly solved by them. First, the GNNs for recommendation address the

cold-start user/item embeddings through optimizing the likelihood of a user adopting an item, which isn't a direct improvement of the embedding quality; second, the GNN model does not specially deal with the cold-start neighbors among all the neighbors when performing the graph convolution. For example in Fig. 1, to represent u_1, the 2-order neighbor u_2 is also a cold-start user who only interacts with i_3 and i_4. The result of graph convolution on the inaccurate embedding of u_2 and the embedding of u_3 together will be propagated to u_1 and hurt its embedding. Existing GNNs ignore the cold-start characteristics of neighbors during the graph convolution process. Although some GNN models such as GrageSAGE [11] or FastGCN [4] filter neighbors before aggregating them, they usually follow a random or an importance sampling strategy, which also ignore the cold-start characteristics of the neighbors. This leads us to the following research problem: *how can we learn more accurate embeddings for cold-start users or items by GNNs?*

Present work. To tackle the above challenges, before performing the GNN model for recommendation, we propose to pre-train the GNN model to enhance the embeddings of the cold-start users or items. Unlike the goal of recommendation, the pre-training task directly reconstructs the cold-start user/item embeddings by mimicking the meta-learning setting via episode based training, as proposed in [34]. Specifically, we pick the users/items with sufficient interactions as the target users/items and learn their ground truth embeddings on the observed abundant interactions. To simulate the real cold-start scenarios, in each training episode, we randomly sample K neighbors for each target user/item, based on which we perform the graph convolution multiple steps to predict the target embedding. The reconstruction loss between the predicted embedding and the ground truth embedding is optimized to directly improve the embedding capacity, making the model easily and rapidly being adapted to new cold-start users/items.

However, the above pre-training strategy still can not explicitly deal with the high-order cold-start neighbors when performing graph convolution. Besides, previous GNN sampling strategies such as random or importance sampling strategies may fail to sample high-order relevant cold-start neighbors due to their sparse interactions. To overcome these challenges, we incorporate a meta aggregator and an adaptive neighbor sampler into the pre-training GNN model. Specifically, the meta aggregator learns cold-start users/items' embeddings on the first-order neighbors by self-attention mechanism under the same meta-learning setting, which is then incorporated into each graph convolution step to enhance the aggregation ability. While the adaptive neighbor sampler is formalized as a hierarchical Markov Sequential Decision Process, which sequentially samples from the low-order neighbors to the high-order neighbors according to the feedbacks provided by the pre-training GNN model. The two components are jointly trained. Since the GNN model can be instantiated by different choices such as the original GCN [19], GAT [31] or FastGCN [4], the proposed pre-training GNN model is model-agnostic. The contributions of this work are as follows:

- We propose a pre-training GNN model to learn high-quality embeddings for cold-start users/items. The model is learned under the meta-learning setting to reconstruct the user/item embeddings, which has the powerful generalization capacity.

- To deal with the cold-start neighbors during the graph convolution process, we further propose a meta aggregator to enhance the aggregation ability of each graph convolution step, and a neighbor sampler to select the effective neighbors adaptively according to the feedbacks of the pre-training GNN model.
- Experiments on both intrinsic embedding evaluation task and extrinsic downstream recommendation task demonstrate the superiority of our proposed pre-training GNN model against the state-of-the-art GNN models.

2 PRELIMINARIES

In this section, we first define the problem and then introduce the graph neural networks that can be used to solve the problem.

We formalize the user-item interaction data for recommendation as a bipartite graph denoted as $G = (U, I, E)$, where $U = \{u_1, \cdots, u_{|U|}\}$ is the set of users and $I = \{i_1, \cdots, i_{|I|}\}$ is the set of items. U and I comprise two types of the nodes in G. Notation $E \subseteq U \times I$ denotes the set of edges that connect the users and items.

We use $\mathcal{N}^l(u)$ to represent the l-order neighbors of user u. When ignoring the superscript, $\mathcal{N}(u)$ indicates the first-order neighbors of u. Similarly, $\mathcal{N}^l(i)$ and $\mathcal{N}(i)$ are defined for items.

Let $f : U \cup V \to \mathbb{R}^d$ be the encoding function that maps the users/items to d-dimension real-valued vectors. We use $\mathbf{h}_u$ and $\mathbf{h}_i$ to denote the embedding of user u and item i respectively. Given a bipartite graph G, we aim to pre-train the encoding function f that is able to be applied on the downstream recommendation task to improve its performance. In the following sections, we mainly take user embedding as an example to explain the proposed model. Item embedding can be explained in the same way.

2.1 GNN for Recommendation

The encoding function f can be instantiated by various GNNs. Take GraphSAGE as an example, we first sample neighbors for each user u randomly and then perform the graph convolution

$$\begin{aligned} \mathbf{h}_{\mathcal{N}(u)}^l &= \text{AGGREGATE}(\{\mathbf{h}_i^{l-1}, \forall i \in \mathcal{N}(u)\}), &(1) \\ \mathbf{h}_u^l &= \sigma(\mathbf{W}^l \cdot \text{CONCAT}(\mathbf{h}_u^{l-1}, \mathbf{h}_{\mathcal{N}(u)}^l)), \end{aligned}$$

to obtain the embedding of u, where l denotes the current convolution step and $\mathbf{h}_u^l$ denotes user u's embedding at this step. Similarly, we can obtain the item embedding $\mathbf{h}_i^l$ at the l-th convolution step. Once the embeddings of the last step L for all the users and items are obtained, we calculate the relevance score $y(u, i) = \mathbf{h}_u^{L\mathsf{T}} \mathbf{h}_i^L$ between user u and item i and adopt the BPR loss [13, 38], i.e.,

$$\mathcal{L}_{BPR} = \sum_{(u,i) \in E, (u,j) \notin E} -\ln \sigma(y(u, i) - y(u, j)) + \lambda ||\Theta_{gnn}||_2^2, \quad (2)$$

to optimize the user preferences over items.

The above presented GNNs are end-to-end models that can learn user/item embeddings and then recommend items to users simultaneously. For addressing the cold-start users/items, the GNNs can incorporate the high-order collaborative signal through iteratively repeating the sampling and the convolution processes. However,
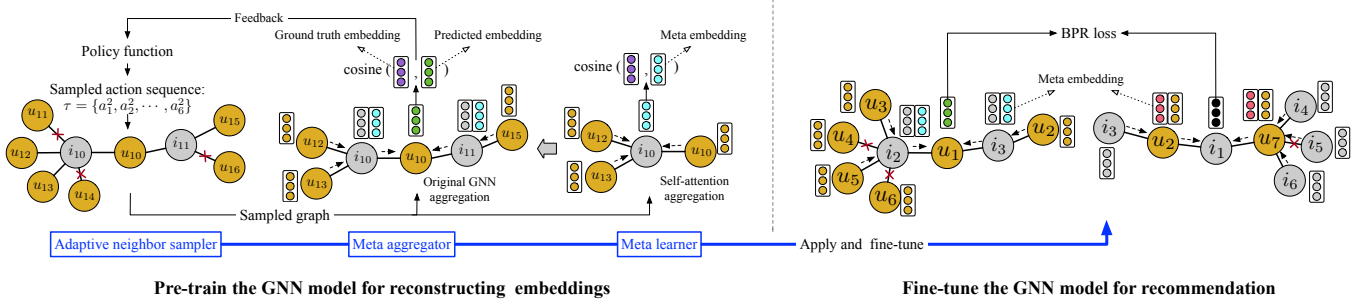
Figure 2: The overall framework of pre-training and fine-tuning the GNN model for recommendation. The pre-training GNN model contains a meta aggregator which has incorporated a self-attention-based meta learner at each step of the original GNN aggregation, and a neighbor sampler which samples the neighbors adaptively according to the feedbacks from the cosine similarity between the predicted embedding and the ground truth embedding. The pre-trained GNN model is applied and fine-tuned on the downstream recommendation task.

the goal of recommendation shown in Eq.(2) can not explicitly improve the embedding quality of the cold-start users/items.

3 THE PRE-TRAINING GNN MODEL

This section introduces the proposed pre-training GNN model to learn the embeddings for the cold-start users and items. We first describe a basic pre-training GNN model, and then explain a meta aggregator and an adaptive neighbor sampler that are incorporated in the model to further improve the embedding performance. Finally we explain how the model is fine-tuned on the downstream recommendation task. The overview framework is shown in Fig. 2.

3.1 The Basic Pre-training GNN Model

We propose a basic pre-training GNN model to reconstruct the cold-start user/item embeddings in the meta-learning setting. To achieve the goal, we need abundant cold-start users/items as the training instances. Since we also need ground truth embeddings of the cold-start users/items to learn f, we simulate those users/items from the target users/items with abundant interactions. The ground truth embedding for each user u, i.e., $\mathbf{h}_u$, is learned upon the observed abundant interactions by NCF[1] [14]. To mimic the cold-start users/items, in each training episode, we randomly sample K neighbors for each target user/item. We repeat the sampling process L-1 steps from the target user to the L-1-order neighbors, which results in at most $K^l (1 \leq l \leq L)$ l-order neighbors for each target user/item. Similar to GraphSAGE [11], we sample high-order neighbors to improve the computational efficiency. Upon the sampled first/high-order neighbors for the target user u, the graph convolution described in Eq. (1) is applied L-1 steps to obtain the embeddings $\{\mathbf{h}_1^{L-1}, \cdots, \mathbf{h}_K^{L-1}\}$ for the K first-order neighbors of u. Then we aggregate them together to obtain the embedding of the target user u. Unlike the previous L-1 steps that concatenates $\mathbf{h}_u^{l-1}$ and $\mathbf{h}_{\mathcal{N}(u)}^l$ to obtain $\mathbf{h}_u^l$ for each neighbor (Cf. Eq. (1)), we only use $\mathbf{h}_{\mathcal{N}(u)}^L$ to represent the target embedding $\mathbf{h}_u^L$, as we aim to predict the target embedding by the neighbors' embeddings:

$$
\begin{aligned}
\mathbf{h}_{\mathcal{N}(u)}^L &= \text{AGGREGATE}(\{\mathbf{h}_i^{L-1}, \forall i \in \mathcal{N}(u)\}), &(3) \\
\mathbf{h}_u^L &= \sigma(\mathbf{W}^L \cdot \mathbf{h}_{\mathcal{N}(u)}^L).
\end{aligned}
$$

Finally, we use cosine similarity to measure the difference between the predicted target embedding $\mathbf{h}_u^L$ and the ground-truth embedding $\mathbf{h}_u$, as proposed by [16], due to its popularity as an indicator for the semantic similarity between embeddings:

$$
\Theta_f^* = \underset{\Theta_f}{\arg\max} \sum_u \cos(\mathbf{h}_u^L, \mathbf{h}_u), \qquad (4)
$$

where $\Theta_f = \{\mathbf{W}^L, \Theta_{gnn}\}$ is the set of the parameters in f.

Training GNNs in the meta-learning setting can explicitly reconstruct the user/item embeddings, making GNNs easily and rapidly being adapted to new cold-start users/items. After the model is trained, for a new arriving cold-start user or item, based on the few first-order neighbors and the high-order neighbors, we can predict an accurate embedding for it. However, the basic pre-training GNN model doesn't specially address the cold-start neighbors. During the original graph convolution process, the inaccurate embeddings of the cold-start neighbors and the embeddings of other neighbors are equally treated and aggregated to represent the target user/item. Although some GNN models such as GrageSAGE or FastGCN filter neighbors before aggregating them, they usually follow the random or importance sampling strategies, which ignore the cold-start characteristics of the neighbors. Out of this consideration, we incorporate a meta aggregator and an adaptive neighbor sampler into the above basic pre-training GNN model.

3.2 Meta Aggregator

We propose the Meta Aggregator to deal with the cold-start neighbors. Suppose the target node is u and one of its neighbor is i, if i is interacted with sparse nodes, its embedding, which is inaccurate, will affect the embedding of u when performing graph convolution by the GNN f. Although the cold-start issue of i is dealt with when

---

[1]The matrix factorization-based model is good enough to learn high-quality user/item embeddings from the abundant interactions.

i acts as another target node, embedding i, which is parallel to embedding u, results in a delayed effect on u' embedding. Thus, before training the GNN f, we train another function g under the similar meta-learning setting as f. The meta learner g learns an additional embedding for each node only based on its first-order neighbors, thus it can quickly adapt to new cold-start nodes and produce more accurate embeddings for them. The embedding produced by g is combined with the original embedding at each convolution in f. Although both f and g are trained under the same meta-learning setting, f is to tackle the cold-start target ndoes, but g is to enhance the cold-start neighbors' embeddings.

Specifically, we instantiate g as a self-attention encoder [30]. For each user u, g accepts the initial embeddings $\{\mathbf{h}_1^0, \cdots, \mathbf{h}_K^0\}$ of the K first-order neighbors for u as input, calculates the attention scores of all the neighbors to each neighbor i of u, aggregates all the neighbors' embeddings according to the attention scores to produce the embedding $\mathbf{h}_i$ for each i, and finally averages the embeddings of all the neighbors to get the embedding $\tilde{\mathbf{h}}_u$, named as the meta embedding of user u. The process is formulated as:

$$
\begin{aligned}
\{\mathbf{h}_1, \cdots, \mathbf{h}_K\} &\leftarrow \text{SELF\_ATTENTION}(\{\mathbf{h}_1^0, \cdots, \mathbf{h}_K^0\}), \quad (5) \\
\tilde{\mathbf{h}}_u &= \text{AVERAGE}(\{\mathbf{h}_1, \cdots, \mathbf{h}_K\}).
\end{aligned}
$$

The self-attention technique, which pushes the dissimilar neighbors further apart and pulls the similar neighbors closer together, can capture the major preference of the nodes from its neighbors. The same cosine similarity described in Eq.(4) is used as the loss function to measure the difference between the predicted meta embedding $\tilde{\mathbf{h}}_u$ and the ground truth embeding $\mathbf{h}_u$. Once g is learned, we add the meta embedding $\tilde{\mathbf{h}}_u$ into each graph convolution step of the GNN f in Eq. (1):

$$
\mathbf{h}_u^l = \sigma(\mathbf{W}^l \cdot \text{CONCAT}(\tilde{\mathbf{h}}_u, \mathbf{h}_u^{l-1}, \mathbf{h}_{\mathcal{N}(u)}^l)), \quad (6)
$$

where the target embedding $\mathbf{h}_u^{l-1}$ of the former step, the aggregated neighbor embedding $\mathbf{h}_{\mathcal{N}(u)}^l$ of this step are learned following the basic pre-training GNN model. For a target user u, Eq. (6) is repeated L-1 steps to obtain the embeddings $\{\mathbf{h}_1^{L-1}, \cdots, \mathbf{h}_K^{L-1}\}$ for its K first-order neighbors, Eq. (3) is also applied on them to get the final embedding $\mathbf{h}_u^L$, and finally the same cosine similarity in Eq. (4) is used to optimize the parameters of the meta aggregator, which includes the parameters Θ_f of the basic pre-training GNN and Θ_g of the meta-learner. The meta aggregator extends the original GNN graph convolution through emphasizing the representations of the cold-start neighbors in each convolution step, which can improve the final embeddings of the target users/items.

3.3 The Adaptive Neighbor Sampler

The proposed sampler does not make any assumption about what kind of neighbors are useful for the target users/items. Instead, it learns an adaptive sampling strategy according to the feedbacks from the pre-training GNN model. To achieve this goal, we cast the task of neighbor sampler as a hierarchical Markov Decision Process (MDP) [28, 47]. Specifically, we formulate the neighbor sampler as $L - 1$ MDP subtasks where the l-th subtask indicates sampling

the l-order neighors. The subtasks are performed sequentially by sampling from the second-order to L-order neighbors[2]. When the l-th subtask deletes all the neighbors or the L-th subtask is finished, the overall task is finished. We will introduce how to design the state, action and the reward for these subtasks as below.

State. The l-th subtask takes an action at the t-th l-order neighbor to determine whether to sample it or not according to the state of the target user u, the formerly selected neighbors, and the t-th l-order neighbor to be determined. We define the state features $\mathbf{s}_t^l$ for the t-th l-order neighbor as the cosine similarity and the element-wise product between its initial embedding and the target user u's initial embedding, the initial embedding of each formerly selected neighbor by the l-1-th subtask and the average embedding of all the formerly selected neighbors respectively.

Action and Policy. We define the action $a_t^l \in \{0, 1\}$ for the t-th l-order neighbor as a binary value to represent whether to sample the neighbor or not. We perform a_t^l by the policy function P:

$$
\begin{aligned}
\mathbf{H}_t^l &= \text{ReLU}(\mathbf{W}_1^l \mathbf{s}_t^l + \mathbf{b}^l), \quad (7) \\
P(a_t^l | \mathbf{s}_t^l, \Theta_s^l) &= a_t^l \sigma(\mathbf{W}_2^l \mathbf{H}_t^l) + (1 - a_t^l)(1 - \sigma(\mathbf{W}_2^l \mathbf{H}_t^l)),
\end{aligned}
$$

where $\mathbf{W}_1^l \in \mathbb{R}^{d_s \times d}$, $\mathbf{W}_2^l \in \mathbb{R}^{d \times 1}$ and $\mathbf{b}^l \in \mathbb{R}^{d_s}$ are the parameters to be learned, d_s is the number of the state features and d is the embedding size. Notation $\mathbf{H}_t^l$ represents the embedding of the input state and $\Theta_s^l = \{\mathbf{W}_1^l, \mathbf{W}_2^l, \mathbf{b}^l\}$. Sigmoid function σ is used to transform the input state into a probability.

Reward. The reward is a signal to indicate whether the performed actions are reasonable or not. Suppose the sampling task is finished at the l'-th subtask, each action of the formerly performed l' subtasks accepts a delayed reward after the last action of the l'-level subtask. In another word, the immediate reward for an action is zero except the last action. The reward is formulated as:

$$
R(a_t^l, \mathbf{s}_t^l) = \begin{cases} \cos(\hat{\mathbf{h}}_u^L, \mathbf{h}_u) - \cos(\mathbf{h}_u^L, \mathbf{h}_u) & \text{if } t = |\mathcal{N}^{l'}(u)| \wedge l = l'; \\ 0 & \text{otherwise}, \end{cases}
$$

(8)

where $\mathbf{h}_u^L$ is the predicted embedding of the target user u after the L-step convolution by Eq. (6) and Eq. (3), while $\hat{\mathbf{h}}_u^L$ is predicted in the same way but on the sampled neighbors following the policy function in Eq. (7). The cosine similarity between the predicted embedding and the ground truth embedding indicates the performance of the pre-training GNN Model. The difference between the performance caused by $\hat{\mathbf{h}}_u^L$ and $\mathbf{h}_u^L$ reflects the sampling effect.

Objective Function. We find the optimal parameters of the policy function defined in Eq. (7) by maximizing the expected reward $\sum_\tau P(\tau; \Theta_s) R(\tau)$, where $\tau = \{s_1^1, a_1^1, s_2^1, \cdots, s_t^{l'}, a_t^{l'}, s_{t+1}^{l'}, \cdots\}$ is a sequence of the sampled actions and the transited states, $P(\tau; \Theta_s)$ denotes the corresponding sampling probability, $R(\tau)$ is the reward for the sampled sequence τ, and $\Theta_s = \{\Theta_s^1, \cdots, \Theta_s^L\}$. Since there are

---

[2]Since the first-order neighbors are crucial for depicting the user profile, we maintain all the first-order neighbors and only sample the high-order neighbors.

Algorithm 1: The Joint Training Process.

Input: $Train_T = \{(u_k, i_k)\}$, the ground truth embeddings $\{(\mathbf{h}_u, \mathbf{h}_i)\}$, a pre-trained meta learner with Θ_g^0, meta aggregator with Θ_f^0 and Θ_g^0 and neighbor sampler with Θ_s^0.

1 Initialize $\Theta_s = \Theta_s^0, \Theta_f = \Theta_f^0, \Theta_g = \Theta_g^0$;
2 **for** *epoch from 1 to E* **do**
3   **foreach** u_k *or* i_k *in* $Train_T$ **do**
4   **for** l *in* $\{2, 3, \cdots, L\}$ **do**
5   Sample a sequence of actions $\tau^l = \{a_1^l, \cdots, a_t^l, \cdots, a_{|\mathcal{N}^l(u)|}^l\}$ by Eq. (7);
6   **if** $\forall\, a_t^l = 0$ *or* $l = L$ **then**
7   Compute $R(a_{|\mathcal{N}^l(u)|}^l, s_{|\mathcal{N}^l(u)|}^l)$ by Eq. (8);
8   Compute gradients by Eq. (9);
9   Break;
10   Update Θ_s;
11   **if** *Jointly Training* **then**
12   Update Θ_g and Θ_f ;

---

too many possible action-state trajectories for the entire sequence, we adopt the monto-carlo policy gradient [39] to sample M action-state trajectories and calculate the gradients:

$$\nabla_{\Theta_s} = \frac{1}{M} \sum_{m=1}^{M} \sum_{l=1}^{l'} \sum_{t=1}^{|\mathcal{N}^l(u)|} \nabla_{\Theta_s} \log P(a_t^{m,l} | s_t^{m,l}, \Theta_s^l) R(a_t^{m,l}, s_t^{m,l}), \quad (9)$$

where $s_t^{m,l}$ represents the state of the t-th l-order neighbor in the m-th action-state trajectory, and $a_t^{m,l}$ denotes the corresponding action. $\mathcal{N}^l(u)$ indicates the set of all the l-order neighbors.

Algorithm 1 shows the training process of the adaptive neighbor sampler. At each step l, we sample a sequence of actions A^l (Line 5). If all the actions at the l-th step equal to zero or the last L-th step is performed (Line 6), the whole task is finished, then we compute the reward (Line 7) and the gradients (Line 8). After an epoch of sampling, we update the parameters of the sampler (Line 10). If it is jointly trained with the meta learner and the meta aggregator, we also update their parameters (Line 12).

3.4 Model Training

The whole process of the pre-training GNN model is shown in Algorithm 2, where we first pre-train the meta learner g only based on first-order neighbors (Line 1), and then incorporate g into each graph convolution step to pre-train the meta aggregator (Line 2), next we pre-train the neighbor sampler with feedbacks from the pre-trained meta aggregator (Line 3), and finally we jointly train the meta learner, the meta aggregator and the neighbor sampler together (Line 4). Same as the settings of [8, 47], to have a stable update during joint training, each parameter $\Theta \in \{\Theta_f, \Theta_g, \Theta_s\}$ is updated by a linear combination of its old version and the new old version, i.e., $\Theta_{new} = \lambda\Theta_{new} + (1 - \lambda)\Theta_{old}$, where $\lambda \ll 1$.

3.5 Downstream Recommendation Task

After the pre-training GNN model is learned, we can fine-tune it in the recommendation downstream task. Specifically, for each target

---

Algorithm 2: The Overall Training Process.

1 Pre-train the meta learner with parameter Θ_g;
2 Pre-train the meta aggregator with parameter Θ_f when fixing Θ_g;
3 Pre-train the neighbor sampler with parameter Θ_s by Algorithm 1 when fixing Θ_g and Θ_f;
4 Jointly train the three modules together with parameters Θ_g, Θ_f and Θ_s by running Algorithm 1;

Table 1: Statistics of the Datasets.

Dataset	#Users	#Items	#Interactions	#Sparse Ratio
MovieLens-1M	6,040	3,706	1,000,209	4.47%
MOOCs	82,535	1,302	458,453	0.42%
Last.fm	992	1,084,866	19,150,868	1.78%

user u and his neighbors $\{\mathcal{N}^1(u), \cdots, \mathcal{N}^L(u)\}$ of different order, we first use the pre-trained neighbor sampler to sample proper high-order neighbors $\{\mathcal{N}^1(u), \hat{\mathcal{N}}^2(u) \cdots, \hat{\mathcal{N}}^L(u)\}$, and then use the pre-trained meta aggregator to produce the user embedding $\mathbf{h}_u^L$. The item embeddings are generated in the same way. Then we transform the embeddings and make a product between a user and an item to obtain the relevance score $y(u, i) = \sigma(\mathbf{W} \cdot \mathbf{h}_u^L)^{\mathrm{T}} \sigma(\mathbf{W} \cdot \mathbf{h}_i^L)$ with parameters $\Theta_r = \{\mathbf{W}\}$. The BPR loss defined in Eq. (2) is used to optimize Θ_r and fine-tune Θ_g, Θ_f and Θ_s.

4 EXPERIMENT

In this section, we present two types of experiments to evaluate the performance of the proposed pre-training GNN model. One is an intrinsic evaluation which aims to directly evaluate the quality of the user/item embedding predicted by the pre-training model. The other one is an extrinsic evaluation which applies the proposed pre-training model into the downstream recommendation task and indirectly evaluate the recommendation performance.

4.1 Experimental Setup

Dataset. We evaluate on three public datasets including MovieLens-1M (Ml-1M)[3] [12], MOOCs[4] [47] and Last.fm[5]. Table 1 illustrates the statistics of these datasets. The code is available now [6].

Baselines. We select three types of baselines including the state-of-the-art neural matrix factorization model, the general GNN models and the special GNN models for recommendation:

- **NCF [14]**: is a neural matrix factorization model which combines Multi-layer Perceptron and matrix factorization to learn the embeddings of users and items.
- **GraphSAGE [11]**: is a general GNN model which samples neighbors randomly and aggregates them by the AVERAGE function.
- **GAT [31]**: is a general GNN model which aggregates neighbors by the attention mechanism without sampling.

- **FastGCN [4]**: is also a general GNN model which samples the neighbors by the important sampling strategy and aggregates neighbors by the same aggregator as GCN [19].
- **FBNE [3]**: is a special GNN model for recommendation, which samples the neighbors by the importance sampling strategy and aggregates them by the AVERAGE function based on the explicit user-item and the implicit user-user/item-item interactions.
- **LightGCN [13]**: is a special GNN model for recommendation, which discards the feature transformation and the nonlinear activation functions in the GCN aggregator.

For each GNN model, we evaluate the corresponding pre-training model. For example, for the GAT model, Basic-GAT means we apply GAT into the basic pre-training GNN model proposed in Section 3.1, Meta-GAT indicates we incorporate the meta aggregator proposed in Section 3.2 into Basic-GAT, NSampler-GAT represents that we incorporate the adaptive neighbor sampler proposed in Section 3.3 into Basic-GAT, and GAT* is the final poposed pre-training GNN model that incorporates both the meta aggregator and the adaptive neighbor sampler into Basic-GAT.

The original GAT and LightGCN models use the whole adjacency matrix, i.e., all the neighbors, in the aggregation function. To train them more efficiently, we implement them in the same sampling way as GraphSAGE, where we randomly sample at most 10 neighbors for each user/item. Then the proposed pre-training GNN model is performed under the sampled graph.

Intrinsic and Extrinsic Settings. We divide each dataset into the meta-training set D_T and the meta-test set D_N. We train and evaluate the pre-training GNN model in the intrinsic user/item embedding inference task on D_T. Once the model is trained, we fine-tune it in the extrinsic downstream recommendation task and evaluate it on D_N. We select the users/items from each dataset with sufficient interactions as the target users/items in D_T, as the intrinsic evaluation needs the true embeddings of users/items inferred from the sufficient interactions. Take the scenario of cold-start users as an example, we divide the users with the number of the direct interacted items more than n_i into D_T and leave the rest users into D_N. We select n_i as 60 and 20 for the dataset Ml-1M and MOOCs respectively. Since the users in Last.fm interact with too many items, we randomly sample 200 users, put 100 users into D_T and leave the rest 100 users into D_N. For each user in D_N, we only keep its K-shot items to simulate the cold-start users. Similarly, for the cold-start item scenario, we divide the items with the number of the direct interacted users more than n_u into D_T and leave the rest items into D_N, where n_u is set as 60, 20 and 15 for MovieLens-1M, MOOCs and Last.fm respectively. In the intrinsic task, K is set as 3 and 8, while in the extrinsic task, K is set as 8. The embedding size d is set as 256. The number of the state features d_s is set as 2819.

4.2 Intrinsic Evaluations: Embedding Inference

In this section, we conduct the intrinsic evaluation of inferring the embeddings of cold-start users/items by the proposed pre-training GNN model. Both the evaluations on the user embedding inference and the item embedding inference are performed.

Training and Test Settings. We use the meta-training set D_T to perform the intrinsic evaluation. Specifically, we randomly split

D_T into the training set $Train_T$ and the test set $Test_T$ with a ratio of 7:3. We train NCF [14] to get the ground-truth embeddings for the target users/items in both $Train_T$ and $Test_T$[7]. To mimic the cold-start users/items on $Test_T$, we randomly keep K neighbors for each user/item, which results in at most K^l neighbors ($1 \leq l \leq 3$) for each target user/item. Thus $Test_T$ is changed into $Test'_T$.

The original GNN models are trained by BPR loss in Eq. (2) on $Train_T$. The proposed pre-training GNN models are trained by the cosine similarity in Eq. (4) on $Train_T$. The NCF model is trained transductively to obtain the user/item embeddings on the merge dataset of $Train_T$ and $Test'_T$. The embeddings in both the proposed models and the GNN models are initialized by the NCF embedding results. We use Spearman correlation [16] to measure the agreement between the ground truth embedding and the predicted embedding.

Overall Performance. Table 2 shows the overall performance of the proposed pre-training GNN model and all the baselines using 3-order neighbors. The results show that compared with the baselines, our proposed pre-training GNN model significantly improves the quality of the user/item embeddings (+33.9%-58.4% in terms of Spearman correlation). Besides, we have the following findings:

- Through incorporating the high-order neighbors, the GNN models can improve the embedding quality of the cold-start users/items compared with the NCF model (+2.6%-24.9% in terms of Spearman correlation).
- All the basic pre-training GNN models beat the corresponding GNN models by improving 1.79-15.20% Spearman correlation, which indicates the basic pre-training GNN model is capable of reconstructing the cold-start user/item embeddings.
- Compared with the basic pre-training GNN model, both the meta aggregator and the adaptive neighbor sampler can improve the embedding quality by 1.09%-18.20% in terms of the Spearman correlation, which indicates that the meta aggregator can indeed strengthen each layer's aggregation ability and the neighbor sampler can filter out the noisy neighbors.
- When the neighbor size K decreases from 8 to 3, the Spearman correlation of all the baselines significantly decrease 0.08%-6.10%, while the proposed models still keep a competitive performance.
- We also investigate the effect of the propagation layer depth L on the model performance. In particular, we set the layer depth L as 1,2,3 and 4, and report the performance in Fig. 3. The results show when L is 3, most algorithms can achieve the best performance, while only using the first-order neighbors performs the worst[8], which implies incorporating proper number of layers can alleviate the cold-start issue.

4.3 Extrinsic Evaluation: Recommendation

In this section, we apply the pre-training GNN model into the downstream recommendation task and evaluate the performance.

Training and Testing Settings. We consider the scenario of the cold-start users and use the meta-test set D_N to perform recommendation. For each user in D_N, we select top 10% of his interacted

---

[7]We concatenate the embeddings produced by both the MLP and the GMF modules in NCF as the ground-truth embedding.
[8]When L is 1, the neighbor sampler and the meta aggregator are not used, thus all the models get the same performance.

Table 2: Overall performance of user/item embedding inference (Spearman correlation). The layer depth L is 3.

Methods	Ml-1M (user)		MOOCs (user)		Last.fm (user)		Ml-1M (item)		MOOCs (item)		Last.fm (item)	
	3-shot	8-shot	3-shot	8-shot	3-shot	8-shot	3-shot	8-shot	3-shot	8-shot	3-shot	8-shot
NCF	-0.017	0.063	-0.098	-0.062	0.042	0.117	-0.118	-0.017	-0.036	0.027	-0.036	-0.018
GraphSAGE	0.035	0.105	0.085	0.128	0.104	0.134	0.113	0.156	0.116	0.182	0.112	0.198
Basic-GraphSAGE	0.076	0.198	0.103	0.152	0.132	0.184	0.145	0.172	0.172	0.196	0.166	0.208
Meta-GraphSAGE	0.258	0.271	0.298	0.320	0.186	0.209	0.434	0.448	0.288	0.258	0.312	0.333
NSampler-GraphSAGE	0.266	0.284	0.294	0.336	0.196	0.212	0.448	0.460	0.286	0.306	0.326	0.336
GraphSAGE*	**0.368**	**0.375**	**0.302**	**0.338**	**0.326**	**0.384**	**0.470**	**0.491**	**0.316**	**0.336**	**0.336**	**0.353**
GAT	0.020	0.049	0.092	0.138	0.092	0.125	0.116	0.126	0.108	0.118	0.106	0.114
Basic-GAT	0.046	0.158	0.104	0.168	0.158	0.180	0.134	0.168	0.112	0.126	0.209	0.243
Meta-GAT	0.224	0.282	0.284	0.288	0.206	0.212	0.438	0.462	0.294	0.308	0.314	0.340
NSampler-GAT	0.296	0.314	0.339	0.354	0.198	0.206	0.464	0.472	0.394	0.396	0.338	0.358
GAT*	**0.365**	**0.379**	**0.306**	**0.366**	**0.309**	**0.394**	**0.496**	**0.536**	**0.362**	**0.384**	**0.346**	**0.364**
FastGCN	0.009	0.012	0.063	0.095	0.082	0.114	0.002	0.036	0.007	0.018	0.007	0.013
Basic-FastGCN	0.082	0.146	0.083	0.146	0.104	0.149	0.088	0.113	0.099	0.121	0.159	0.182
Meta-FastGCN	0.181	0.192	0.282	0.280	0.224	0.274	0.216	0.266	0.248	0.278	0.230	0.258
NSampler-FastGCN	0.188	0.194	0.281	0.286	0.226	0.277	0.268	0.288	0.267	0.296	0.246	0.253
FastGCN*	**0.198**	**0.212**	**0.288**	**0.291**	**0.266**	**0.282**	**0.282**	**0.298**	**0.296**	**0.302**	**0.268**	**0.278**
FBNE	0.034	0.102	0.053	0.065	0.142	0.164	0.168	0.190	0.137	0.168	0.127	0.133
Basic-FBNE	0.162	0.190	0.162	0.185	0.135	0.180	0.176	0.209	0.157	0.180	0.167	0.173
Meta-FBNE	0.186	0.204	0.269	0.284	0.175	0.192	0.426	0.449	0.236	0.272	0.178	0.182
NSampler-FBNE	0.208	0.216	0.259	0.283	0.203	0.207	0.422	0.439	0.226	0.273	0.164	0.183
FBNE*	**0.242**	**0.265**	**0.306**	**0.321**	**0.206**	**0.219**	**0.481**	**0.490**	**0.301**	**0.382**	**0.182**	**0.199**
LightGCN	0.093	0.108	0.060	0.068	0.162	0.184	0.201	0.262	0.181	0.232	0.213	0.245
Basic-LightGCN	0.178	0.192	0.212	0.226	0.182	0.192	0.318	0.336	0.234	0.260	0.252	0.290
Meta-LightGCN	0.226	0.241	0.272	0.285	0.206	0.221	0.336	0.346	0.314	0.331	0.372	0.392
NSampler-LightGCN	0.238	0.256	0.286	0.294	0.204	0.212	0.348	0.384	0.296	0.314	0.356	0.401
LightGCN*	**0.270**	**0.286**	**0.292**	**0.309**	**0.229**	**0.234**	**0.382**	**0.408**	**0.334**	**0.353**	**0.386**	**0.403**

(a) MovieLens (user)   (b) MovieLens (item)   (c) Moocs (user)   (d) Moocs (item)   (e) Last.fm (user)   (f) Last.fm (item)
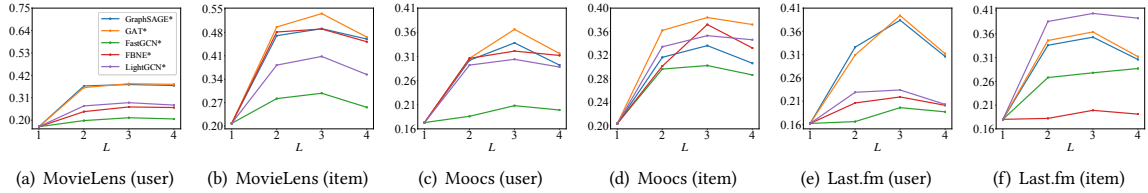
Figure 3: Performance of user/item embedding inference under different layer depth L when K=3.

items in chronological order into the training set $Train_N$, and leave the rest items into the test set $Test_N$. We pre-train our model on D_T and fine-tune it on $Train_N$ according to Section 3.5.

The original GNN and the NCF models are trained by the BPR loss function in Eq. (2) on D_T and $Train_N$. For each user in $Test_N$, we calculate the user's relevance score to each of the rest 90% items. We adopt Recall@$\mathcal{K}$ and NDCG@$\mathcal{K}$ as the metrics to evaluate the items ranked by the relevance scores. By default, we set $\mathcal{K}$ as 20 for Ml-1m and Moocs. For Last.fm, since there are too many items, we set $\mathcal{K}$ as 200.

Overall Performance. Table 3 shows the overall recommendation performance. The results indicate that the proposed basic pre-training GNN models outperform the corresponding original GNN

models by 0.40%-3.50% in terms of NDCG, which demonstrates the effectiveness of the basic pre-training GNN model on the cold-start recommendation performance. Upon the basic pre-training model, adding the meta aggregator and the adaptive neighbor sampler can further improve 0.30%-6.50% NDCG respectively, which indicates the two components can indeed alleviate the impact caused by the cold-start neighbors when embedding the target users/items, thus they can improve the downstream recommendation performance.

Case Study. We attempt to understand how the proposed pre-training model samples the high-order neighbors of the cold-start users/items by the MOOCs dataset. Fig. 4 illustrates two sampling cases, where notation * indicates the users/items are cold-start. The cold-start item i^*_{1099} is "Corporate Finance", which only interacts

Table 3: The Overall Recommendation Performance.

Methods	Ml-1M		MOOCs		Last.fm	
	Recall	NDCG	Recall	NDCG	Recall	NDCG
NCF	0.008	0.101	0.021	0.047	0.005	0.007
GraphSAGE	0.006	0.082	0.085	0.066	0.003	0.011
Basic-GraphSAGE	0.013	0.135	0.082	0.091	0.007	0.044
Meta-GraphSAGE	0.016	0.209	0.096	0.116	0.007	0.097
NSampler-GraphSAGE	0.021	0.221	0.101	0.122	0.008	0.088
GraphSAGE*	**0.024**	**0.235**	**0.110**	**0.129**	**0.008**	**0.131**
GAT	0.008	0.099	0.023	0.055	0.006	0.033
Basic-GAT	0.016	0.163	0.032	0.093	0.006	0.147
Meta-GAT	0.017	0.191	0.063	0.123	0.009	0.184
NSampler-GAT	0.012	0.188	0.084	0.132	0.010	0.199
GAT*	**0.014**	**0.208**	**0.100**	**0.139**	**0.018**	**0.232**
FastGCN	0.003	0.019	0.064	0.089	0.006	0.068
Basic-FastGCN	0.008	0.102	0.099	0.117	0.012	0.083
Meta-FastGCN	0.009	0.123	0.105	0.124	0.018	0.116
NSampler-FastGCN	0.011	0.118	0.108	0.128	0.020	0.136
FastGCN*	**0.012**	**0.123**	**0.119**	**0.140**	**0.023**	**0.186**
FBNE	0.002	0.088	0.048	0.041	0.009	0.013
Basic-FBNE	0.009	0.104	0.064	0.087	0.003	0.032
Meta-FBNE	0.012	0.101	0.088	0.101	0.006	0.087
NSampler-FBNE	0.013	0.118	0.102	0.117	0.006	0.099
FBNE*	**0.014**	**0.121**	**0.117**	**0.138**	**0.007**	**0.129**
LightGCN	0.014	0.207	0.102	0.112	0.001	0.083
Basic-LightGCN	0.012	0.211	0.112	0.121	0.005	0.097
Meta-LightGCN	0.018	0.221	0.120	0.139	0.005	0.101
NSampler-LightGCN	0.020	0.218	0.116	0.132	0.007	0.106
LightGCN*	**0.022**	**0.227**	**0.123**	**0.142**	**0.007**	**0.114**
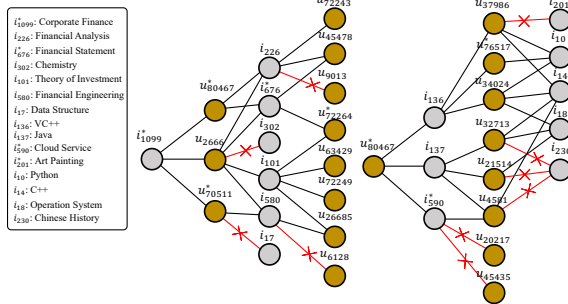
Figure 4: Case study of the adaptive neighbor sampling.

with three users. Our proposed neighbor sampler samples a second-order item i_{676}^*, "Financial Statement". Although i_{676}^* only interacts with two users, it is relevant to the target item i_{1099}^*. Similarly, the cold-start user u_{80467}^*, who likes computer science, only selects three computer science related courses. The proposed neighbor sampler samples a second-order user u_{76517}^*. Although it only interacts with two courses, they are "Python" and "VC++" which are relevant to computer science. However, the importance-based sampling strategies in FastGCN and FBNE cannot sample these neighbors, as they are cold-start with few interactions.

5 RELATED WORK

Cold-start Recommendation. Cold-start issue is a fundamental challenge in recommender systems. On one hand, existing recommender systems incorporate the side information such as spatial information [40, 44], social trust path [10, 36, 41, 42] and knowledge graphs [35, 37] to enhance the representations of the cold-start users/items. However, the side information is not always available, making it intractable to improve the cold-start embedding's quality. On the other hand, researchers solve the cold-start issue by only mining the underlying patterns behind the user-item interactions. One kind of the methods is meta-learning [9, 23, 26, 34], which consists of metric-based recommendation [29] and model-based recommendation [7, 20, 22, 24]. However, few of them capture the high-order interactions. Another kind of method is GNNs, which leverage user-item bipartite graph to capture high-order collaborative signals for recommendation. The representative models include Pinsage [45], NGCF [38], LightGCN [13], FBNE [3] and CAGR [42]. Generally, the recommendation-oriented GNNs optimize the likelihood of a user adopting an item, which isn't a direct improvement of the embedding quality of the cold-start users or items.

Pre-training GNNs. Recent advances on pre-training GNNs aim to empower GNNs to capture the structural and semantic properties of an input graph, so that it can easily generalize to any downstream tasks with a few fine-tuning steps on the graphs [17]. The basic idea is to design a domain specific pretext task to provide additional supervision for exploiting the graph structures and semantic properties. Examples include 1) graph-level pretext task, which either distinguishes subgraphs of a certain node from those of other vertices [25] or maximize the mutual information between the local node representation and the global graph representations[27, 32]. 2) Node-level task, which perform node feature and edge generation [17] pretext tasks. 3) Hybrid-level task, which considers both node and graph-level tasks [15, 46]. However, none of these models explore pre-training GNNs for recommendation, and we are the first to study the problem and define the reconstruction of the cold-start user/item embeddings as the pretext task.

6 CONCLUSION

This work explores pre-training a GNN model for addressing the cold-start recommendation problem. We propose a pretext task as reconstructing cold-start user/item embeddings to explicitly improve their embedding quality. We further incorporate a self-attention-based meta aggregator to improve the aggregation ability of each graph convolution step, and propose a sampling strategy to adaptively sample neighbors according to the GNN performance. Experiments on three datasets demonstrate the effectiveness of our proposed pre-training GNN against the original GNN models. We will explore multiple pretext tasks in the future work.

ACKNOWLEDGMENTS

REFERENCES

[1] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. 2014. Spectral Networks and Locally Connected Networks on Graphs. In *ICLR'14*.

[2] Hongxu Chen, Hongzhi Yin, Tong Chen, Quoc Viet Hung Nguyen, Wen-Chih Peng, and Xue Li. 2019. Exploiting Centrality Information with Graph Convolutions for Network Representation Learning. In *ICDE'19*. 590–601.

[3] Hongxu Chen, Hongzhi Yin, Tong Chen, Weiqing Wang, Xue Li, and Xia Hu. 2020. Social Boosted Recommendation with Folded Bipartite Network Embedding. *IEEE Transactions on Knowledge and Data Engineering* (2020).

[4] Jie Chen, Tengfei Ma, and Cao Xiao. 2018. FastGCN: Fast Learning with Graph Convolutional Networks via Importance Sampling. In *ICLR'18*.

[5] Jianfei Chen, Jun Zhu, and Le Song. 2018. Stochastic Training of Graph Convolutional Networks with Variance Reduction. In *ICML'18*, Vol. 80. 941–949.

[6] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering. In *NeurIPS'16*. 3837–3845.

[7] Zhengxiao Du, Xiaowei Wang, Hongxia Yang, Jingren Zhou, and Jie Tang. 2019. Sequential Scenario-Specific Meta Learner for Online Recommendation. In *SIGKDD'19*. 2895–2904.

[8] Jun Feng, Minlie Huang, Li Zhao, Yang Yang, and Xiaoyan Zhu. 2018. Reinforcement Learning for Relation Classification From Noisy Data. In *AAAI'18*. 5779–5786.

[9] Chelsea Finn, Pieter Abbeel, and Sergey Levine. 2017. Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks. In *ICML'17*, Vol. 70. 1126–1135.

[10] Zhabiz Gharibshah, Xingquan Zhu, Arthur Hainline, and Michael Conway. 2020. Deep Learning for User Interest and Response Prediction in Online Display Advertising. *Data Sci. Eng.* 5, 1 (2020), 12–26.

[11] William L. Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. In *NeurIPS'17*. 1024–1034.

[12] F. Maxwell Harper and Joseph A. Konstan. 2016. The MovieLens Datasets: History and Context. *ACM Trans. Interact. Intell. Syst.* (2016), 19:1–19:19.

[13] Xiangnan He, Kuan Deng, Xiang Wang, Yan Li, Yongdong Zhang, and Meng Wang. 2020. LightGCN: Simplifying and Powering Graph Convolution Network for Recommendation. In *SIGIR'20*.

[14] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural Collaborative Filtering. In *WWW'17*. 173–182.

[15] Weihua Hu, Bowen Liu, Joseph Gomes, Marinka Zitnik, Percy Liang, Vijay S. Pande, and Jure Leskovec. 2020. Strategies for Pre-training Graph Neural Networks. In *ICLR'20*.

[16] Ziniu Hu, Ting Chen, Kai-Wei Chang, and Yizhou Sun. 2019. Few-Shot Representation Learning for Out-Of-Vocabulary Words. In *ACL'19*. 4102–4112.

[17] Ziniu Hu, Yuxiao Dong, Kuansan Wang, Kai-Wei Chang, and Yizhou Sun. 2020. GPT-GNN: Generative Pre-Training of Graph Neural Networks. In *SIGKDD'20*.

[18] Wenbing Huang, Tong Zhang, Yu Rong, and Junzhou Huang. 2018. Adaptive Sampling Towards Fast Graph Representation Learning. In *NeurIPS'18*. 4563–4572.

[19] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *ICLR'17*.

[20] Hoyeop Lee, Jinbae Im, Seongwon Jang, Hyunsouk Cho, and Sehee Chung. 2019. MeLU: Meta-Learned User Preference Estimator for Cold-Start Recommendation. In *SIGKDD'19*. 1073–1082.

[21] Greg Linden, Brent Smith, and Jeremy York. 2003. Amazon.com Recommendations: Item-to-Item Collaborative Filtering. *IEEE Internet Comput.* (2003), 76–80.

[22] Yuanfu Lu, Yuan Fang, and Chuan Shi. 2020. Meta-learning on heterogeneous information networks for cold-start recommendation. (2020).

[23] Tsendsuren Munkhdalai and Hong Yu. 2017. Meta Networks. In *ICML'17 (Proceedings of Machine Learning Research, Vol. 70)*, Doina Precup and Yee Whye Teh (Eds.). 2554–2563.

[24] Feiyang Pan, Shuokai Li, Xiang Ao, Pingzhong Tang, and Qing He. 2019. Warm Up Cold-start Advertisements: Improving CTR Predictions via Learning to Learn ID Embeddings. In *SIGIR'19*. 695–704.

[25] Jiezhong Qiu, Qibin Chen, Yuxiao Dong, Jing Zhang, Hongxia Yang, Ming Ding, Kuansan Wang, and Jie Tang. 2020. GCC: Graph Contrastive Coding for Graph Neural Network Pre-Training. In *SIGKDD'20*.

[26] Jake Snell, Kevin Swersky, and Richard S. Zemel. 2017. Prototypical Networks for Few-shot Learning. In *NeurIPS'17*. 4077–4087.

[27] Fan-Yun Sun, Jordan Hoffmann, Vikas Verma, and Jian Tang. 2020. InfoGraph: Unsupervised and Semi-supervised Graph-Level Representation Learning via Mutual Information Maximization. In *ICLR'20*.

[28] Ryuichi Takanobu, Tianyang Zhang, Jiexi Liu, and Minlie Huang. 2019. A Hierarchical Framework for Relation Extraction with Reinforcement Learning. In *AAAI'19*. 7072–7079.

[29] Manasi Vartak, Arvind Thiagarajan, Conrado Miranda, Jeshua Bratman, and Hugo Larochelle. 2017. A Meta-Learning Perspective on Cold-Start Recommendations for Items. In *NeurIPS'17*. 6904–6914.

[30] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *NeurIPS'17*. 5998–6008.

[31] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. In *ICLR'18*.

[32] Petar Velickovic, William Fedus, William L Hamilton, Pietro Liò, Yoshua Bengio, and R Devon Hjelm. 2019. Deep Graph Infomax. In *ICLR'19*.

[33] Ricardo Vilalta and Youssef Drissi. 2002. A perspective view and survey of meta-learning. *Artificial intelligence review* 18, 2 (2002), 77–95.

[34] Oriol Vinyals, Charles Blundell, Tim Lillicrap, Koray Kavukcuoglu, and Daan Wierstra. 2016. Matching Networks for One Shot Learning. In *NeurIPS'16*. 3630–3638.

[35] Hongwei Wang, Fuzheng Zhang, Miao Zhao, Wenjie Li, Xing Xie, and Minyi Guo. 2019. Multi-Task Feature Learning for Knowledge Graph Enhanced Recommendation. In *WWW'19*. 2000–2010.

[36] Qinyong Wang, Hongzhi Yin, Hao Wang, Quoc Viet Hung Nguyen, Zi Huang, and Lizhen Cui. 2019. Enhancing Collaborative Filtering with Generative Augmentation. In *SIGKDD'19*. ACM, 548–556.

[37] Xiang Wang, Xiangnan He, Yixin Cao, Meng Liu, and Tat-Seng Chua. 2019. KGAT: Knowledge Graph Attention Network for Recommendation. In *SIGKDD'19*. 950–958.

[38] Xiang Wang, Xiangnan He, Meng Wang, Fuli Feng, and Tat-Seng Chua. 2019. Neural Graph Collaborative Filtering. In *SIGIR'19*. 165–174.

[39] Ronald J. Williams. 1992. Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. *Mach. Learn.* 8, 229–256.

[40] Hongzhi Yin, Bin Cui, Yizhou Sun, Zhiting Hu, and Ling Chen. 2014. LCARS: A Spatial Item Recommender System. *TOIS'14* 32, 3 (2014), 11:1–11:37.

[41] Hongzhi Yin, Qinyong Wang, Kai Zheng, Zhixu Li, Jiali Yang, and Xiaofang Zhou. 2019. Social Influence-Based Group Representation Learning for Group Recommendation. In *ICDE'19*. IEEE, 566–577.

[42] Hongzhi Yin, Qinyong Wang, Kai Zheng, Zhixu Li, and Xiaofang Zhou. 2020. Overcoming Data Sparsity in Group Recommendation. *TKDE'20* abs/2010.00813 (2020).

[43] Hongzhi Yin, Weiqing Wang, Hao Wang, Ling Chen, and Xiaofang Zhou. 2017. Spatial-Aware Hierarchical Collaborative Deep Learning for POI Recommendation. *TKDE'19* 11 (2017), 2537–2551.

[44] Hongzhi Yin, Weiqing Wang, Hao Wang, Ling Chen, and Xiaofang Zhou. 2017. Spatial-Aware Hierarchical Collaborative Deep Learning for POI Recommendation. *IEEE Trans. Knowl. Data Eng.* 29, 11 (2017), 2537–2551.

[45] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L. Hamilton, and Jure Leskovec. 2018. Graph Convolutional Neural Networks for Web-Scale Recommender Systems. In *SIGKDD'18*. 974–983.

[46] Yuning You, Tianlong Chen, Zhangyang Wang, and Yang Shen. 2020. When Does Self-Supervision Help Graph Convolutional Networks?. In *ICML'20*.

[47] Jing Zhang, Bowen Hao, Bo Chen, Cuiping Li, Hong Chen, and Jimeng Sun. 2019. Hierarchical Reinforcement Learning for Course Recommendation in MOOCs. In *AAAI'19*. 435–442.