ApeGNN: Node-Wise Adaptive Aggregation in GNNs for Recommendation

Dan Zhang
Tsinghua University
zd21@mails.tsinghua.edu.cn

Yifan Zhu
Tsinghua University
zhuyifan@tsinghua.edu.cn

Yuxiao Dong
Tsinghua University
yuxiaod@tsinghua.edu.cn

Yuandong Wang
Tsinghua University
wangyd21@tsinghua.edu.cn

Wenzheng Feng
Tsinghua University
wenzhengfeng96@gmail.com

Evgeny Kharlamov
Bosch Center for Artificial Intelligence
evgeny.kharlamov@de.bosch.com

Jie Tang*
Tsinghua University
jietang@tsinghua.edu.cn

Abstract

In recent years, graph neural networks (GNNs) have made great progress in recommendation. The core mechanism of GNNs-based recommender system is to iteratively aggregate neighboring information on the user-item interaction graph. However, existing GNNs treat users and items equally and cannot distinguish diverse local patterns of each node, which makes them suboptimal in the recommendation scenario. To resolve this challenge, we present a node-wise adaptive graph neural network framework ApeGNN. ApeGNN develops a node-wise adaptive diffusion mechanism for information aggregation, in which each node is enabled to adaptively decide its diffusion weights based on the local structure (e.g., degree). We perform experiments on six widely-used recommendation datasets. The experimental results show that the proposed ApeGNN is superior to the most advanced GNN-based recommender methods (up to 48.94%), demonstrating the effectiveness of node-wise adaptive aggregation.

CCS Concepts

• **Information systems** → Recommender systems.

Keywords

Recommender Systems; Graph Neural Networks; Node-wise Adaptive Aggregation

ACM Reference Format:
Dan Zhang, Yifan Zhu, Yuxiao Dong, Yuandong Wang, Wenzheng Feng, Evegeny Kharlamov, Jie Tang. 2023. ApeGNN: Node-Wise Adaptive Aggregation in GNNs for Recommendation. In *Proceedings of the ACM Web Conference 2023 (WWW '23), April 30–May 04, 2023, Austin, TX, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3543507.3583530

*Jie Tang is the corresponding author.

The code is available at https://github.com/zhangdan0602/ApeGNN.

WWW '23, April 30–May 04, 2023, Austin, TX, USA

ACM ISBN 978-1-4503-9416-1/23/04.
https://doi.org/10.1145/3543507.3583530

1 Introduction

Graph neural networks (GNNs) have produced remarkable performance on the task of recommendation [9, 13, 35, 42, 45]. Specifically, GNNs perform the message passing process over graph structures to aggregate local neighborhood information and stack multiple representation layers in high-order propagation [8, 17, 34]. GNNs-based recommender systems apply the information propagation operation over the user-item bipartite graphs, such as NGCF [36], LightGCN [9], and MixGCF [13].

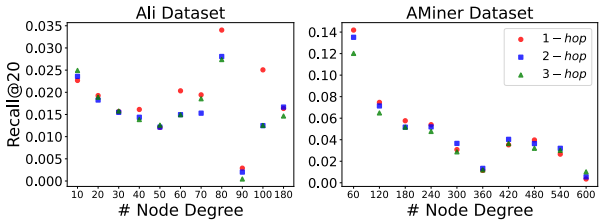

Figure 1: Local structures of users are diverse and node-wise aggregation is important in GNNs-based recommendation scenario.

However, there are remaining issues in GNNs-based models for recommendation. **Firstly, node types are not distinguished in GNNs-based recommendation.** User-item interaction network is a special type of graph in which edges can only be existed in the middle of users and items. In other words, there is no direct communication between two users or items. Up to now, the existing GNNs-based recommendation method is no different from the general GNNs that handles the other normal graphs, such as reference graphs [17], social networks etc. , that is, the same modeling strategies are adopted for all users and items. Generally, user aggregates information from (only) 1-hop item neighbors with 0-th layer embeddings, from 2-hop users with 1-th layer representations, and from 3-hop items again with 2-th layer embeddings, in a high-order neighborhood aggregation setting. By taking all layers the same, the aggregation step in GNNs-based recommender systems ignores the semantic differences between users and items at each subgraph structure—and by extension their embeddings. **Secondly, local structures of different users/items are diverse in recommendation.** In Figure 1, we show a motivating example to fully understand motivation of this work, i.e., the node-wise necessity for GNNs-based recommendation. To be specific, we perform 1-/2-/3-hop aggregation and propagation with LightGCN on Ali

and AMiner datasets respectively, and show the relation between the Recall@20 result and user nodes with different degree intervals. We find that average result distribution of different degree nodes on each hop propagation is very unbalanced and the optimal propagation number for each node is uncertain, which means each user/item has its local structure. This demonstrates that different nodes with different structures in recommendation should be given different importance. Take these issues into account, we argue that node-wise adaptive aggregation in GNNs for recommendation is a necessary step to learn the node-level difference among layers. Therefore, we propose to study whether each node should be treated differently in different layers during the aggregation process of GNNs-based recommendation methods.

Table 1: Critical comparison between existing GNNs-based models for recommendation.

Models	Explicit Degree Info.	Linear Propagation	Node-wise
GAT [33]	✗	✗	✓
LightGCN [9]	✓	✓	✗
ADC [47]	✓	✓	✗
ApeGNN	✓	✓	✓

However, the aggregation step in previous GNNs-based recommender systems treats all nodes as the same and ignores their variant importance [38], which results in sub-optimal performance [24]. As Table 1 shows, we compare existing representative GNNs-based models for recommendation. Previous attention-based GNN models (e.g., GAT) attempt to model the different influences among user and item nodes without considering the local diversity [1, 3, 10, 33, 41]. Moreover, for differentiating the importance of interacted historical items, the degree-based GNNs process (e.g., LightGCN) can improve embedding learning and achieve better performance [9] in bipartite graph for recommendation [9, 13, 36], but still ignores the local diversity issue. In recent diffusion-based GNNs studies (e.g., ADC), appropriate neighborhoods are selected flexibly to enhance GNNs' expressiveness during the propagation [19, 47]. For instance, the graph diffusion convolution [19] model incorporates the heat kernel and node centrality information into GNNs' message passing process to reinforce structural smoothness. Nevertheless, GDC is based on the assumption of homophily and does not perform well on the link prediction task. An adaptive graph convolution work (ADC) [47] proposes to leverage the heat kernel theory [39] to learn the optimal neighborhood for enhancing the low-frequency filters. Though the heat kernel is originally used to enable neighborhood selection in GNNs, it also provides a natural solution to address the aforementioned issues for recommendation.

In this work, we present a novel **A**da**P**tiv**E** model (ApeGNN), which conducts node-wise adaptive aggregation in GNNs for recommendation. Instead of treating each user and item equally at each layer during the high-order aggregation and propagation, we leverage the graph diffusion process to adaptively assign a unique weight (inner-layer weight) to each hop of neighbors and distinguish the information from different GNNs layers, promoting the development of aggregation methods from fixed aggregation to node-wise aggregation. The idea of ApeGNN for GNNs-based recommendation is that each user u_i and each item v_j respectively have a unique aggregation weight $\theta(t(u_i), l)$ and $\theta(t(v_j), l)$ with coefficient t and layer l. This diffusion weights represent the distinctive contributions of u_i's and v_j's embedding that are captured by ApeGNN at each layer.

ApeGNN can serve as a plug-in and be naturally incorporated into any existing GNNs-based models for recommendation without modifying the architecture of models. We conduct extensive experiments on six widely-used public datasets and compare ApeGNN with representative GNNs-based and attention-based models to demonstrate the effectiveness of ApeGNN. We follow the same experimental procedure—data splits, optimization, and evaluation—as existing GNNs-based recommendation studies [9, 36]. The results suggest that the proposed ApeGNN can consistently outperform state-of-the-art GNN baselines (relative improvement up to 48.94% on Ali, 24.09% on Amazon and 7.67% on AMiner) among all data sets on both Recall and NDCG, which shows the benefits of node-based adaptive aggregation. Case studies show that adaptive node weights contribute to the overall and each layer's performance.

2 Graph Neural Networks for Recommendation

In this section, we revisit GNNs-based recommendation systems and discuss the limitations of existing GNNs for recommendation.

2.1 Preliminaries

Generally, the input of a GNNs-based recommendation model is a user-item bipartite graph $\mathcal{G} = \{\mathcal{U}, \mathcal{V}, \mathcal{R}\}$ with the sets of users $\mathcal{U} = \{u_1, u_2, \ldots, u_m\}$, items $\mathcal{V} = \{v_1, v_2, \ldots, v_n\}$, and the interactions such as purchase, rate and click, among users and items $\mathcal{R} \in \mathbb{R}^{m \times n}$, where m is the amount of users and n denotes the total number of items. For each interaction, $r_{(u_i v_j)} \in \mathcal{R}$ is set to 1 if u_i interacts with v_j and 0 otherwise. With $\mathcal{R}$, the purpose of the recommendation system is to predict the items which the user will interact further. GNNs have been recently found wide adoption in recommender systems [5, 40, 42, 48]. Similar to common GNN models, GNN-based recommender systems perform message passing over the input graph structure to get contextual representations. Commonly, the message passing process consists of the aggregation and pooling.

Aggregation. For node u, the general representation at l-layer during propagation in GCN-based models can be represented as:

$$\mathbf{h}_u^{(l)} = U(\mathbf{h}_u^{(l-1)}, AGG(\{\mathbf{h}_v^{(l-1)}, \forall v \in \mathcal{N}_u)\}), \tag{1}$$

where AGG and U stand for aggregation and update function respectively. Take GCN [17] for example, it aggregates each node's neighborhood nodes and performs message passing. In particular, the propagation rule Eq.(1) of GCN [17] is defined by:

$$\mathbf{h}_u^{(l)} = \sigma(\mathbf{W}^{(l)}(\mathbf{h}_u^{(l-1)} + \mathbf{T}\mathbf{h}_v^{(l-1)})) \tag{2}$$

where $\mathbf{h}_u^{(l)}$ denotes the embedding representation of u after l-layer GCN. In particular, $\mathbf{h}_u^{(0)}$ is the 0-layer embedding; $\sigma(\cdot)$ is non-linear activation for feature propagation, $\mathbf{W}$ is a weight matrix; $\mathcal{N}_u$ is the neighborhood of node u. $\mathbf{T}$ can be calculated by $\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}$, where $\mathbf{D}$ is the diagonal node degree matrix with $\mathbf{D}_{ii} = \sum_j A_{ij}$. Note that $\tilde{\mathbf{A}} = \mathbf{I} + \mathbf{A}$ and $\tilde{\mathbf{D}} = \mathbf{I} + \mathbf{D}$, in which $\mathbf{A}$ is the adjacency matrix with self-loop and $\mathbf{I}$ is the identity matrix that implies self-loop connections on nodes.

Inspired by the design and idea of the GCN model, emerging studies (e.g., GCMC [32] and NGCF [36]) attempt to adopt GCN structure for recommendation tasks. However, recent studies [2, 9]

reported that some of the most used designs of GCNs do not contribute much to recommendation performance, which complicates learning process. They are designed to simplify the GCN for recommendation and achieve better performance. Taking LightGCN [9] as an example, it deletes $\sigma(\cdot)$, $\mathbf{W}$ and $\mathbf{I}$. Thus, its aggregation and propagation process are expressed as:

$$\mathbf{h}_u^{(l)} = \mathbf{h}_u^{(l-1)} + \mathbf{T}\mathbf{h}_v^{(l-1)}, \tag{3}$$

where normalized item $\mathbf{T}$ is computed by $\mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$.

Pooling. In GCN [17], the representation vector learned by the final layer is utilized to perform node classification. However, the final representation of nodes could be varying by changing the pooling operation at the each GCN layer (also called the layer combination operation). In most GNNs-based recommender models, the pooling function combines the embeddings $(\mathbf{h}_u^{(0)}, \mathbf{h}_u^{(1)}, \cdots, \mathbf{h}_u^{(L)})$ propagated at each layer and generates the final representation vector $\mathbf{h}_u^*$. Common pooling operations include weighted sum-based [9, 23, 31], concat-based pooling [30, 36], attention-based and other approaches [45]. Generally, the pooling function p of user u can be formulated as:

$$\mathbf{h}_u^* = p(\mathbf{h}_u^{(0)}, \mathbf{h}_u^{(1)}, \cdots, \mathbf{h}_u^{(L)}), \tag{4}$$

where L is the total number of propagation layers and the pooling function of item v is similar to that of user. In particular, weighted sum pooling function is the most commonly used. For example, $\mathbf{h}_u^* = \sum_{l=0}^{L} \beta_l \mathbf{h}_u^{(l)}$, β_l is the weight of l-th layer embedding for the final embedding. Especially, they assign a constant value (e.g., $\beta_l = 1/(L+1)$) as fixed weight for the embeddings of the each layer to perform layer combination. In other words, the importance of the embeddings represented by each subgraph structure in different GCN layers is the same.

2.2 The limitation of GNNs-based models

By default, the current GNNs-based recommendation models have the following setup in the aggregation process: First, these methods treat all nodes in the bipartite graph as the same; Second, they consider the embeddings of multiple layers with the same importance in a local perspective.

By design, aggregators of GNNs mainly include two categories: degree-based aggregator represented by LightGCN and attention-based aggregator represented by GAT. In fact, they do not explicitly differentiate nodes with global perspective during the message passing process, while all nodes are in nature different in recommender systems [9, 23, 29].

• **Degree-based Aggregator Represented by LightGCN.** By extending Eq. (3), the graph convolution on the first and second layers for user embedding u_i is formulated as Eq. (5), and item embedding is formulated similarly.

$$\begin{aligned}
\mathbf{h}_{u_i}^{(1)} &= \sum_{v_j \in \mathcal{N}_{u_i}} \frac{1}{(\sqrt{|\mathcal{N}_{u_i}||\mathcal{N}_{u_j}|})} \mathbf{h}_{v_j}^{(0)}, \\
\mathbf{h}_{u_i}^{(2)} &= \sum_{v_j \in \mathcal{N}_{u_i}} \frac{1}{|\mathcal{N}_{v_j}|} \sum_{u_k \in \mathcal{N}_{v_j}} \frac{1}{(\sqrt{|\mathcal{N}_{u_i}||\mathcal{N}_{u_k}|})} \mathbf{h}_{u_k}^{(0)},
\end{aligned} \tag{5}$$

where $1/|\mathcal{N}_{v_j}|$ and $1/(\sqrt{|\mathcal{N}_{u_i}||\mathcal{N}_{u_k}|})$ are the non-sysmmetric and symmetric normalization, respectively. It is observed that the aggregation and propagation process of embeddings for users and items are exactly the same, that is, the information updated from same type (i.e., users or items) nodes is propagated in odd GCN layers $(1, 3, \cdots)$, and messages from the other type is propagated in even GCN layers.

The degree-based aggregator is applied to each node to control the decay factor during aggregation and propagation, potentially making these processes different. However, the normalization coefficient is a constant value computed by degrees of node, which is irrelevant with node types and the layer subgraph structures. In fact, users and items are two different types of nodes with distinctive semantics in a user-item bipartite graph, and their own embeddings in different subgraphs should be naturally considered dissimilar as well. This effect has been initially investigated by grouping users in higher-order GCN layers [23]. In this work, we first attempt to explicitly differentiate node-aware users and items in the simple GNN architecture and examine whether different treatments to them can benefit GNNs-based recommender systems.

• **Attention-based Aggregator Represented by GAT.** When representation of a node is obtained through all neighbors, the contributions of these neighbors should be different as well. Different from degree-based aggregator, attention-based aggregator assigns different weights for neighbors of each node during aggregation in local graph structure. The aggregation process is described as:

$$\mathbf{h}_{u_i}^{(l)} = \sigma(\mathbf{W} \cdot \alpha_{u_i v_j} \cdot \mathbf{h}_{v_j}^{(l-1)} + \mathbf{b}), \forall v_j \in \mathcal{N}_{u_i}. \tag{6}$$

where $\alpha_{u_i v_j}$ denotes attentive scores and it is usually calculated by Softmax function. The attention weights are calculated in a local graph and propagated implicitly to next layer via non-linear activation function, which ignores decay factor of each node's centrality distribution with a node-wise perspective. However, the embeddings at different layers of (graph) neural networks are in nature supposed to capture different levels of features [37, 46]. In addition, as introduced above, the information updated from same type nodes and the other type in GNNs-based recommendation models is propagated in odd and even GCN layers, respectively. In this paper, we study whether the different weights on each node should be assigned from one layer to another. Therefore, in this study, we try to address this research problem: "to what extent such node-wise and layer-wise differentiation can improve the performance of GNNs for recommendation?"

3 The ApeGNN

We design a GNNs-based recommender system ApeGNN in this section. The main idea is to differentiate each user as well as item from nodes during aggregation and calculate the impacts in a node-wise way in GNNs. To achieve this, we incorporate the idea of graph diffusion based adaptive operations into the aggregation.

3.1 Node-Wise Adaptive Aggregation in GNNs

To incorporate node importance into existing aggregation in GNNs-based models for recommendation, we design a node-wise adaptive aggregation mechanism. For a user u_i and its neighborhood nodes $\mathcal{N}_{u_i}$, the aggregation function AGG with weight coefficient function

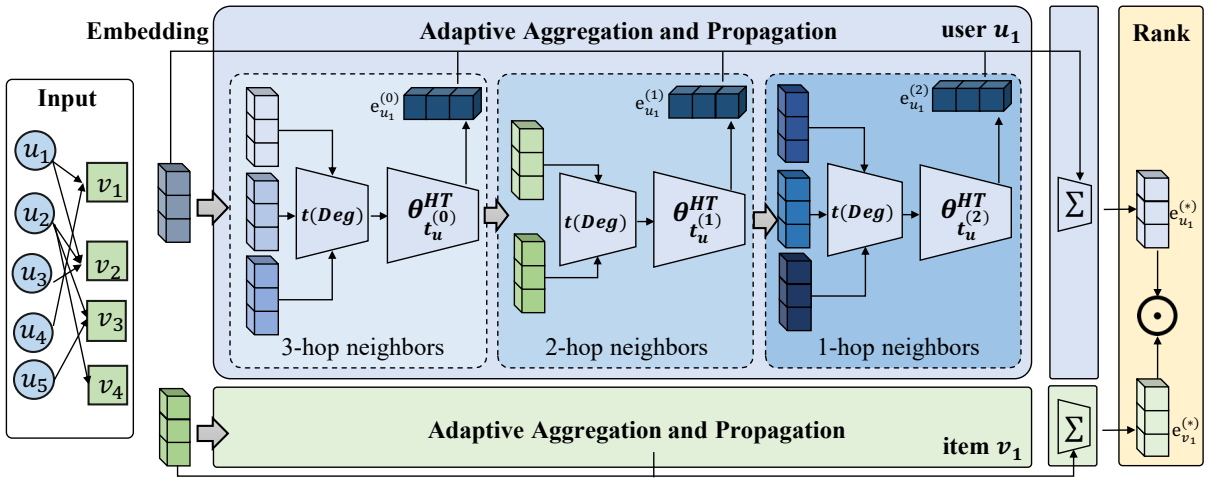

Figure 2: The overview of the ApeGNN model when $L = 3$. In ApeGNN, the users and items at each layer are modeled differently with adaptive weights θ via the heat kernel (default) and PPR in the process of aggregation and propagation.

$\theta(t_{u_i})$ for user u_i can be presented as:

$$\mathbf{h}_{u_i} = AGG(\{\mathbf{h}_{v_j}, \forall v_j \in \mathcal{N}_{u_i}\}; \theta(t_{u_i})). \tag{7}$$

where t_{u_i} is a parameter of adaptive operation for user u_i. Similarly, the aggregation function AGG with weight coefficient function $\theta(t_{v_j})$ for item v_j can be described in Eq. (8). Then, we introduce how to use $\theta(t_{u_i})$ and $\theta(t_{v_j})$ during aggregation process.

$$\mathbf{h}_{v_j} = AGG(\{\mathbf{h}_{u_i}, \forall u_i \in \mathcal{N}_{v_j}\}; \theta(t_{v_j})), \tag{8}$$

Weighting Coefficients θ. As previously discussed, semantics contained in embeddings at different layers tend to be different in (graph) neural networks. For GNNs over a bipartite graph, each layer is made up of either users or items, which naturally makes the semantics of each layer differently. In other words, the embeddings at different layers should be treated distinctly by setting different weights to capture the unique semantics of each layer during aggregation. Specially, we propose two methods, namely heat kernel (HT) [39] and personalized PageRank (PPR) [25], to simulate the graph diffusion process and provide better importance selection support. Here, we will introduce how to formulate these two methods as coefficients ($\theta^{HT}_{(t^{(l)}_{u_i})}, \theta^{PPR}_{(t^{(l)}_{u_i})}$) in detail.

• **The heat kernel.** The feature propagation between nodes in the GNNs-based models can be viewed as the practice of Newton's law of cooling (also known as the heat kernel) [20, 47], in which heat is transferred from regions with higher temperature to regions with lower one. That is, the embedding propagation between two nodes is naturally proportional to their representation. Thus, the derivation of this prior knowledge is calculated as:

$$\frac{\partial \mathbf{h}_{u_i}(t)}{\partial t} + \Delta \mathbf{h}_{u_i}(t) = 0, \tag{9}$$

$$\frac{\mathrm{d}\mathbf{h}_{u_i}(t)}{\mathrm{d}t} = -\sum_{v_j \in \mathcal{N}_{u_i}} \mathbf{A}_{u_i v_j}(\mathbf{h}_{u_i}(t) - \mathbf{h}_{v_j}(t)), \tag{10}$$

where $\mathbf{h}_{u_i}(t)$ and $\mathbf{h}_{v_j}(t)$ denote representations of user u_i and item v_j after time t. This derivation indicates that $\mathbf{h}_{u_i}$ is related to its neighbors at one time. Inspired by diffusion design of the heat kernel, we can incorporate the heat kernel into the GNNs-based models for personalized recommendation. Given an initial definition that the heat kernel can be expressed as $H_t = e^{-t\Gamma}$ at time t in a graph, where Γ ($\Gamma = \mathbf{I} - \mathbf{D}$) is the Laplacian matrix of graph $\mathcal{G}$. According to this definition, Eq. (3) can be reformulated as:

$$\mathbf{h}(t) = H_t \mathbf{h}(0), \tag{11}$$

where $\mathbf{h}(t)$ denotes the hidden representation of a node after the diffusion time t. For $\forall t \geq 0$, the convolution kernel of $\mathcal{G}$ with the heat kernel is formulated as:

$$e^{-t\Gamma}\mathbf{h} = \sum_{l=0}^{\infty} \frac{t^l e^{-t}}{l!} \mathbf{T}^l \mathbf{h}, \tag{12}$$

where l denotes the current layer, the parameter t^l is related to neighborhood size of a node at l-th layer. Therefore, a function $\theta^{HT}_{u_i}$ with t_{u_i} and $\theta^{HT}_{v_j}$ with t_{v_j} for each node are used to automatically learn and update t_u and t_v during training. Its formulation can be described as:

$$\theta^{HT}_{(t^{(l)}_{u_i})} = \frac{t^l_{u_i} e^{-t_{u_i}}}{l!}. \tag{13}$$

• **PPR.** ApeGNN aims to recognize the local pattern from each node's graph structure. To achieve this objective, we utilize the PPR used in PageRank [25] and APPNP [18] to build graph structure information and assign unique weights for each node. The coefficient $\theta^{PPR}_{u_i}$ with teleport probability $t^{(0)}_{u_i} \in (0, 1)$ for user can be represented as:

$$\theta^{PPR}_{(t^{(l)}_{u_i})} = t^{(l)}_{u_i}(1 - t^{(l)}_{u_i})^l. \tag{14}$$

The common points between ApeGNN and APPNP are that we connect GCN with personalized PageRank to propagate long range and reduce the oversmoothing risk, and leverage teleport probability properly to retain the initial features to gain better performance. By leveraging the heat kernel and personalized PageRank, we can assign appropriate weights flexibly for each node to enhance low-frequency filters and enforce smoothness on a graph.

Centrality Importance t. As Figure 1 depicts, the best propagation of u is not certain. Thus, the neighbors weights at each layer should be considered when modeling the importance of nodes, and the different influences among user and item nodes should be modeled when obtaining the representation of a node during the aggregation. Previous research [25] about node importance estimation suggests that the importance of nodes is positively correlated with its centrality in a graph. Generally, the in-degree $D(u_i)$ of user node u_i denotes its centrality and popularity. Therefore, we use in-degree $D(u_i)$ as weights $t_{u_i}^{(0)}$ and $t_{v_j}^{(0)}$ at the initial layer for u_i and v_j to model the differences of users as well as items. Here, we define t for user u_i and obtain a small value for t. The centrality importance of items are similar to users' in Eq. (15).

$$t_{u_i}^{(0)} = \varphi(D(u_i)) = \sigma(log(D(u_i) + \epsilon)), \tag{15}$$

where ϵ is a small positive constant chosen as 10^{-7} and the teleport propability $t_{u_i}^{(0)} \in (0, 1)$.

In ApeGNN, by giving a user-item interaction bipartite graph as the input, the embedding of each user and each item is differentiated via an adaptive aggregation, and this embedding is parameterized with node-wise to form the final representations. The architecture of ApeGNN for recommendation is presented in Figure 2 which illustrates main parts of the model—the node-wise inner-layer aggregation and inter-layer propagation.

3.2 Propagation Process

By using enhanced representation in the aggregation process, we can add each embedding layer to propagation layers to mine higher-order connectivity information. The propagated embeddings of user u_i and item v_j at the l-th layer are formulated as:

$$\begin{aligned} \mathbf{h}_{u_i}^{(l)} &= \theta(t_{u_i}^{(l)}) \sum_{v_j \in \mathcal{N}_{u_i}} p_{(u_i v_j)} \mathbf{h}_{v_j}^{(l-1)}, \\ \mathbf{h}_{v_j}^{(l)} &= \theta(t_{v_j}^{(l)}) \sum_{u_i \in \mathcal{N}_{v_j}} p_{(v_j u_i)} \mathbf{h}_{u_i}^{(l-1)}. \end{aligned} \tag{16}$$

where $t_{u_i}^{(l)}$ and $t_{v_j}^{(l)}$ are the unique weight for u_i as well as v_j at l-th layer, and $p_{(u_i v_j)} = 1/(\sqrt{|\mathcal{N}_{u_i}||\mathcal{N}_{v_j}|})$ is the symmetric normalization suggested by GCN [17]. Note that the effect of self-loop connections in the GCN layer could be captured by a weighted sum as pointed out by LightGCN[9]. By following the same setup, we remove the self-loop connections to reduce the information redundancy.

By extending the embedding aggregation and propagation function with the convolution kernel, the adaptive graph convolution matrix $\mathbf{E}_u$ and $\mathbf{E}_v$ with weight matrix Θ for users and items in ApeGNN at the l-th layer can be represented as Eq. (17):

$$\begin{aligned} \mathbf{E}_u^{(l)} &= \sum_{l=0}^{L} \Theta_{t_u}^{(l)} \mathbf{T}^l \mathbf{E}_u^{(l-1)}, \\ \mathbf{E}_v^{(l)} &= \sum_{l=0}^{L} \Theta_{t_v}^{(l)} \mathbf{T}^l \mathbf{E}_v^{(l-1)}. \end{aligned} \tag{17}$$

By transforming Eq. (17) into the actual model training process, we take the graph convolution matrix $(\mathbf{E}^{(0)}, \cdots, \mathbf{E}^{(L)})$ related to the layer into the pooling operation.

3.3 Pooling

Given the input user and item embeddings $\mathbf{h}_{u_i}^{(0)}$ and $\mathbf{h}_{v_j}^{(0)}$ at the 0-th layer, they are first propagated through higher layers by Eq. (16). Then we update the integration of users as well as items through the embedding of the latest layer and the current layer by Eq. (1). Finally, the embeddings of each layer should be combined to formalize the final embeddings of user u_i and item v_j that are used for final recommendation. In ApeGNN, we formulate Eq. (4) to model the hop-wise semantic differences that each layer of embeddings represents. Thus, the final embedding of u_i and v_j are pooled as:

$$\mathbf{h}_{u_i}^{*} = \sum_{l=0}^{L} \mathbf{h}_{u_i}^{(l)}, \ \mathbf{h}_{v_j}^{*} = \sum_{l=0}^{L} \mathbf{h}_{v_j}^{(l)} \tag{18}$$

where $h_{u_i}^{(l)}$ and $h_{v_j}^{(l)}$ have been used to model the weights and importance of the l-th layer embeddings of user u_i and item v_j, respectively. Eventually, the inner product of embeddings between u_i and v_j is calculated as $\hat{r}_{(u_i,v_j)} = \mathbf{h}_{u_i}^{*} \odot \mathbf{h}_{v_j}^{*}$, which can be exploited to calculate the preference of user u_i on item v_j.

3.4 Optimization

Similar to many other GNNs-based recommendation methods [9, 30, 36], we use the Bayesian Personalized Ranking (BPR) loss [26] to optimize the neural network structure of ApeGNN. The BPR is a pairwise loss which is formalized as:

$$\mathcal{L} = - \sum_{(u_i,v_j,v_k) \in O} \ln \sigma(\hat{r}_{u_i,v_j} - \hat{r}_{u_i,v_k}) + \lambda ||\mathbf{E}^{(0)}||^2, \tag{19}$$

where $O = \{(u_i, v_j, v_k) | (u_i, v_j) \in O^+, (u_i, v_k) \in O^-\}$ are the pairwise training data; O^- is the set of pairs that interactions between the user and the item are not actually observed, while O^+ is the set of interacted records. λ influences the strength of L_2 regularization; sigmoid function is denoted by $\sigma(\cdot)$ in this formulation; and the L_2 regularization is exploited to avoid overfitting issue.

The adaptive operation does not change the GNN nature and flexibility of ApeGNN. In fact, ApeGNN can serve as a plugin into any GNNs-based models for personalized recommendation. Therefore, other advanced training techniques, e.g., the hop-mixing negative sampling strategy [13], can be straightforwardly applied into ApeGNN. In summary, the overall training process is formally described in Algorithm 1 of Appendix.

3.5 Model Analysis

Here, we provide an additional discussion on the differences between ApeGNN and other related models, regarding the designs of the inner-layer aggregation operations.

ApeGNN vs. LightGCN. LightGCN [9] is representative graph convolution network with degree normalization based graph structure, while it ignores the distinctive semantics of each node. ApeGNN considers centrality-based importance to explicitly differentiate each node. Compare to LightGCN, results in Tabel 2 and Table 4 both show the effectiveness of ApeGNN.

ApeGNN vs. GAT. It is obvious that ApeGNN is similar to (dual) attention-based models which perform local aggregation for each layer and consider importance of nodes. Specially, attention-based

models assign weights for local neighbors via node features from a local perspective, while ApeGNN aggregates neighbors adaptively in a node-wise manner via centrality degree from the global perspective. In addition, by executing high-order propagation in attention-based models, the information of previous layer is implicitly propagated via non-linear activation to next layer. The adaptive aggregation of ApeGNN considers the influence of subgraph structure and explicitly propagates the message via a layer coefficient. To sufficiently show that adaptive aggregation is a better choice than attention modules, we provide compared experimental results in Table 3. As it shows, ApeGNN incorporates node-and-layer-specific weights to achieve adaptive aggregation, leading to better performance compared to attention-based methods.

ApeGNN vs. ADC. The diffusion model of ADC is able to learn the scale of neighbors during the propagation of each layer from the network for node classification. In particular, ADC learns the initial value t of every feature channel as well as layer, but ignores the of node importance. In contrast, the adaptive aggregator of ApeGNN learns centrality-aware t for each node which considers the importance of nodes. To prove the effectiveness of ApeGNN, we compare ADC (i.e., trained-t) and ApeGNN in Table 3 in our ablation study. The experiments show that the effectiveness of centrality-based weights in ApeGNN.

3.6 Complexity Analysis

Let the number of nodes and edges be $|\mathcal{U}+\mathcal{V}|$ and $|\mathcal{E}|$, e denote the number of epochs, where d is the embedding size, and L is the layer number of the proposed GNN. The time and memory complexity are summarized as follows.

Time Complexity. We analyze whether and how the adaptive operations in aggregation impact the complexity of GNNs-based recommendation methods. The time complexity of ApeGNN mainly comes from three parts: adjacency matrix construction, adaptive graph convolution operation, and BPR loss calculation. Here, we mainly introduce the time complexity of adaptive graph convolution operation. For the graph convolution module, although additional adaptive operations are integrated into its aggregation, it does not increase its complexity, i.e., $T = O(eL|\mathcal{U}+\mathcal{V}|d)$. Although we try to learn the weighting coefficients as embeddings with fixed size, the cost of all steps is linearly increasing with the T, Thus, the cost of all three steps is linearly changing with the number of e and d—$O(|\mathcal{E}| + ed(|\mathcal{E}| + L|\mathcal{U}+\mathcal{V}|))$, making the time complexity of ApeGNN be equivalent to common GNNs-based recommender systems, such as LightGCN [9]. Take AMiner dataset as an example, the training time of ApeGNN/LightGCN is ~21s/~14s per epoch, which demonstrates that the additional node-wise parameters have limited affects on running time.

Memory Complexity. In adaptive aggregation process , we assign centrality-based weights for each node. Thus, we train models with up to $|\mathcal{U}+\mathcal{V}|$ parameters. Although this approach brings additional memory consumption (e.g., 12.9 GB on Amazon), ApeGNN can achieve significant performance gains with this operation.

4 Experiments

4.1 Experimental Settings

Baselines. For performance comparison, we select various state-of-the-art baselines including MF-based (BPRMF [26], NeuMF [11]), first-order (Mult-VAE [22], GF-CF [28]), high-order (NGCF [36], LightGCN [9], i.e., GNNs-based ones), attention-based (NAIS [10], SASRec [15], GC-SAN [44] and LightSANs [4]) models. We will testify the effectiveness of our model ApeGNN on these four categories. The detailed description of these models is presented in Appedix A.2.

Datasets. To evaluate the effectiveness of ApeGNN, we perform experiments on six public datasets (Ali, Amazon, AMiner, Gowalla, MovieLens, and Yelp2018) collected from real-world platforms with different data sizes, densities and conditions (# Users $\gg$ # Items and # Users $\ll$ # Items) for comparing with representative models. We further testify three datasets (Epinions, ML-1M and Pinterest) which are often evaluated among attention-based models. The detailed meta information of these datasets is shown in Table 5 of Appendix.

Evaluation Metrics. We select two widely-used evaluation metrics, namely Recall and Normalized Discounted Cumulative Gain for the top-K prediction, which is denoted as Recall@K, NDCG@K, to evaluate performance of representative models. Besides, MRR@K and Hit Ratio (HR@K) are further used to evaluate the performance between the ApeGNN and attention-based models. The detailed explanation and calculation of these metrics are presented in Appendix A.3. In the test stage, we select the all items which has not been interacted by this user and regard them as the negative samples, while the interacted items are regarded as positive ones. For each u_i, we rank all predicted items ordered by the preference score $\hat{r}_{(u_i,v_j)}$. By selecting top-K items, we present the average value of metrics on all users in the whole test set. Here, we conduct our experiments for the value of K in the range {5, 10, 20, 50}, and report the results of $K = 20$ for each user for simplicity. The trend of results on others K is similar to $K = 20$. To express convenience, Recall@20 is simplified to Recall/R in Table 2/Table 4, etc.

4.2 Performance Comparison

The overall top-20 performance results on representative models with average Recall and NDCG metrics are presented in Table 2. In Table 2, we highlight the best results in **bold** and second best results in underline. The comparison results between ApeGNN and attention-based models are summarized in Table 3. The improvement (%Improv.) is computed according to the second best results. Based on these results, we obtain the following observations:

Comparison with MF-based models. From Table 2, we observe that high-order models outperform BPR-MF and NeuMF across all datasets. This illustrates they gain significant superiority comparing to traditional deep MF-based models. Further, gathering information from high-order neighbors improves the effectiveness of representation learning. This result is consistent with the claim in existing studies [9, 36]. Particularly, ApeGNN derives significantly better Recall@20 and NDCG@20 than MF-based models on Ali and Amazon datasets.

Table 2: Overall performance (% is omitted) comparison with representative model on six datasets.

Dataset	Ali		Amazon		AMiner		Gowalla		MovieLens		Yelp2018	
Metrics	Recall	NDCG	Recall	NDCG	Recall	NDCG	Recall	NDCG	Recall	NDCG	Recall	NDCG
BPR-MF	1.03	0.46	1.65	0.76	18.42	9.38	14.17	12.04	8.20	10.75	4.72	3.84
NeuMF	0.87	0.37	1.51	0.58	16.84	8.65	13.97	11.63	7.05	9.31	3.88	3.14
Mult-VAE	2.60	0.78	2.30	0.82	17.89	9.81	15.49	11.98	6.18	5.23	5.94	4.64
GF-CF	4.20	1.84	2.38	1.08	18.77	9.81	17.80	14.61	8.23	11.52	6.32	5.16
NGCF	4.26	1.97	2.94	1.23	17.66	9.11	14.22	11.88	9.31	11.37	5.77	4.69
LightGCN	6.13	2.86	4.11	1.86	19.69	9.87	17.75	15.22	9.41	11.36	6.61	5.39
ApeGNN_HK	8.80	4.29	4.98	**2.36**	**21.20**	**10.69**	**18.32**	**15.35**	9.73	11.91	**6.75**	**5.56**
%Improv.	43.56%	50.00%	21.17%	26.88%	7.67%	8.31%	3.21%	0.85%	3.40%	4.84	7.48%	7.96%
ApeGNN_PPR	**9.13**	**4.41**	**5.10**	2.33	20.88	10.29	17.88	15.17	**9.77**	**12.09**	6.59	5.44
%Improv.	48.94%	54.20%	24.09%	25.27%	6.04%	4.26%	0.73%	-	3.83%	6.43%	-	0.93%

Table 3: Performance (% is omitted) comparison between ApeGNN and attention-based models.

Dataset	Models	NAIS	SASRec	GC-SAN	LightSANs	**ApeGNN**
Epinions	Recall@20	1.11	1.45	1.56	1.68	**1.82**
	MRR@20	0.47	0.33	0.44	0.49	**0.91**
	NDCG@20	0.61	0.57	0.69	0.75	**1.11**
	HR@20	1.11	1.45	1.56	1.68	**1.84**
ML-1M	Recall@20	24.75	22.59	15.27	33.31	**26.91**
	MRR@20	42.65	4.4	3.00	8.17	**44.52**
	NDCG@20	24.98	8.29	5.59	13.64	**26.66**
	HR@20	84.57	22.59	15.27	33.31	**86.37**
Pinterest	Recall@20	11.68	9.41	9.64	9.96	**16.66**
	MRR@20	6.16	1.85	1.89	1.98	**9.94**
	NDCG@20	6.06	3.44	3.53	3.60	**9.37**
	HR@20	22.53	9.41	9.64	9.96	**30.46**

Comparison with first-order models. As high-order models, ApeGNN, NGCF and LightGCN overall outperform Mult-VAE and GF-CF on all four datasets, which demonstrates their advantages comparing with first-order models. On Ali dataset, ApeGNN is higher than Mult-VAE in Recall@20 and NDCG@20. Moreover, NGCF and LightGCN underperform GF-CF on Gowalla, but outperform it on Amazon dataset, which may be caused by the different densities of datasets (0.084% in Gowalla v.s. 0.014% in Amazon). Meanwhile, our proposed ApeGNN exceeds GF-CF on all datasets and has an improvement up to 117% (Recall@20 on Ali dataset). These observations show that high-order approaches like ApeGNN have remarkable advantages in extreme sparse scenarios than first-order models.

Comparison with other higher-order models. ApeGNN outperforms LightGCN on all six datasets, by 0.73-48.94% and 0.85-54.20% in terms of Recall@20 and NDCG@20 on average. This testifies that our ApeGNN can better handle deep multi-hop neighbors than LightGCN, which verifies the effectiveness of incorporating the adaptive aggregation function in ApeGNN. It captures the interactions between users and items and distinguishes different importance of embeddings from different layers in aggregation process. Especially, the improvements of ApeGNN on Ali and Amazon dataset are more significant than those on the other datasets. In other words, ApeGNN can learn the representation better for users and items even if the graph is very sparse.

Comparison with attention-based models. As Table 3 shows, LightSANs exploits self-attention network-based recommendation, and it performs better than NAIS and SASRec. This shows the usefulness of modeling the node-wise importance in recommender models. ApeGNN can achieve the optimal performance over the three datasets. Compared to NAIS and LightSANs, ApeGNN utilizes the centrality degree to model the importance of users and items, which helps to differentiate each node. In addition, unlike NAIS, SASRec and LightSANs that use attention mechanism to learn the representations, ApeGNN models high-order user-item interaction to obtain embeddings in GNNs.

4.3 Study of ApeGNN

To comprehensively understand the structure of ApeGNN and investigate the reasons of its effectiveness, further exploration experiments are performed. Firstly, we study the influence of value of t. Secondly, we explore the effect of different layer numbers L in adaptive aggregation module. Finally, we evaluate the influence of regularization L_2 during training.

Effect of value t. To get a better understanding of the proposed ApeGNN model, we evaluate the key components of ApeGNN, i.e. aggregation mechanisms. There are three different values of weight t during aggregation, including fixing t to the initialization value for user type and item type (fixed-t), training one t for each node and item without centrality (trained-t), training a unique t for each user and item with centrality (ApeGNN-t). Here, we compare ApeGNN with its three variants: fixed-t, trained-t and ApeGNN-t. Figure 3 shows the Recall@20 results of ApeGNN on six datasets when the inner-layer weights with t_u, t_v of users and items are set to different values. From the results, we have the main observations as follows.

Overall, we reach the optimal performance when considering centrality of a node and training a unique t for this node on all datasets. It shows that in-degree of nodes is important to represent their centrality and popularity in user-item interacted graph, further verifies that importance should be considered during aggregation and propagation process. Besides, we find that the result of training one t for all nodes and all items respectively is better than fixing t for all nodes on six datasets. It justifies our assumption that users and items are nodes of different types that we should differentiate them to learn better representations and improve the performance of recommendation. To sum up, ApeGNN can differentiate users and items, and assign centrality-based and layer-wise weights via

ApeGNN-t for nodes in aggregation process, which can boost the recommendation performance.

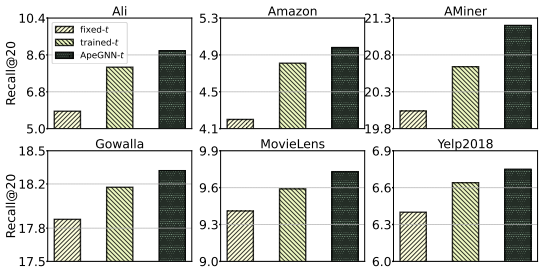

Figure 3: Recall@20 (with omitting %) on six datasets when t is set with different values.

Effect of layer numbers L in propagation. We analyze the effect of layer number of propagation from 1 to 4, provide a detailed comparison with the LightGCN, and highlight the best results with underline and **bold** for LightGCN and ApeGNN, respectively. From Figure 3, we also have the following main findings. Overall, the ApeGNN outperforms LightGCN under all layer settings across all datasets. The result is consistent with Table 2. In general, with the increase of the layer number, the performance first increases and then decreases. For details, the performance increases to peak value with the layer number from 1 to 2 (or 3), then the performance degrades. It indicates that it is helpful for a node to consider its low-order neighbors which can smooth its embedding and obtain powerful representation.

Table 4: Results on top-20 recommendation between LightGCN and ApeGNN on three datasets at different layers.

Dataset		Ali		Amazon		AMiner	
#Layers	Method	Recall	NDCG	Recall	NDCG	Recall	NDCG
1 Layer	LightGCN	4.90	2.19	3.64	1.55	19.09	9.43
	ApeGNN	8.70	4.23	**4.98**	**2.36**	20.90	10.49
2 Layers	LightGCN	2.93	1.30	3.88	1.77	19.69	9.87
	ApeGNN	8.69	4.22	4.91	2.28	**21.20**	**10.69**
3 Layers	LightGCN	5.89	2.77	3.98	1.76	19.24	9.56
	ApeGNN	**8.80**	**4.29**	4.78	2.20	21.04	10.65
4 Layers	LightGCN	6.13	2.86	4.11	1.86	19.38	9.83
	ApeGNN	8.76	4.29	4.70	2.17	21.02	10.61

Effect of coefficient λ of regularization L_2 . In ApeGNN, λ of L_2 is the additional hyper-parameters. As shown in Figure 4, we analyze the influence of different coefficient λ on representative datasets (Amazon and MovieLens). We observe that ApeGNN is relatively insensitive to these hyper-parameters. The optimal value for Amazon and MovieLens both are 10^{-2}. When λ is smaller than 10^{-3}, the performance starts to drop, which indicates that regularization can prevent overfitting in some degree for ApeGNN.

5 Related Work

5.1 Attention-based GNN Models

Attention-based models can enhance the embedding vectors of users and items by applying attention-based mechanism to obtain the interactions on the user-item bipartite graph. To exploit the importance relation between user and item, NAIS [10] automatically learns the importance weight of each interacted item. Similar to

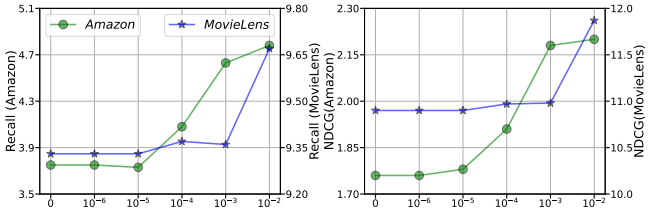

Figure 4: Performance of ApeGNN at the 3-th layer on Amazon and MovieLens datasets.

NAIS, GraphRec [3] models the different contributions of historical items to shape user's interest in a user-item graph. SASRec [15] proposes a self-attention model to capture long-term sequences and semantics, further predicting next action from users' historical items. Inspired by the power of the attention mechanism, LightSANs [4] adopts low-rank decomposed self-attention for next-item recommendation. In contrast, the performance can be improved by projecting interacted items into a GNNs-based model to capture high-order neighbors.

5.2 GNNs-based Recommendation Methods

GNNs-based recommendation approaches model each user and each item as embeddings and learn the embedding vectors by reconstructing the historical behavior between users and items. Early MF-based models (such as PMF [27] and SVD++ [21]) are based on decomposing the interaction adjacency matrix between users and items, and predict users' preferences for unseen items based on these decomposed vectors, which are effective but insufficient to model complex user behaviors and large data inputs. Similar to classical CF methods, the neural collaborative filtering (NCF)-based methods (such as [7, 11, 12, 14, 22, 43]) expanded the inner product of MF in the manner of neural collaborative filtering to increase the capacity, but still faced the problem that it was difficult to learn high-order structural information in the data.

Emerging Graph Convolution Networks (GCNs) put up prominent performance by modeling graph structure and learning representation. Motivated by the efficiency of graph convolution, the GNNs-based paradigm is introduced into recommendation. Early studies such as GCN [17], PinSage [45] and GAT [34] aggregate the neighborhood representations to generate the embeddings of the target node on the spatial domain. Recent studies [9, 32, 36] build a user-item bipartite graph and use GCN to capture CF signals in high-order neighbors on a graph. However, theses GNNs-based models fail to handle different importances of nodes during aggregation.

6 Conclusion

In this paper, we propose ApeGNN which introduces the graph diffusion process into GNNs-based recommendation. ApeGNN addresses the issues that neighborhood types are not adaptive to identify and the importance of each node which is not divided by expanding the propagation of neighborhood. We assign different weights to entities with different types and importance of different nodes, and assign different importance on multi-order neighborhoods. By performing experiments on public recommendation, we empirically show the superiority of the ApeGNN compared with existing baseline models.

Acknowledgments

This research was supported by Natural Science Foundation of China (NSFC) 62276148 and 61836013, Tsinghua-Bosch Joint ML Center and Zhipu.AI.

References

[1] Jingyuan Chen, Hanwang Zhang, Xiangnan He, Liqiang Nie, Wei Liu, and Tat-Seng Chua. 2017. Attentive collaborative filtering: Multimedia recommendation with item-and component-level attention. In *Proceedings of the 40th International ACM SIGIR conference on Research and Development in Information Retrieval*. 335–344.

[2] Lei Chen, Le Wu, Richang Hong, Kun Zhang, and Meng Wang. 2020. Revisiting Graph Based Collaborative Filtering: A Linear Residual Graph Convolutional Network Approach. In *AAAI 2020*. 27–34.

[3] Wenqi Fan, Yao Ma, Qing Li, Yuan He, Eric Zhao, Jiliang Tang, and Dawei Yin. 2019. Graph neural networks for social recommendation. In *The world wide web conference*. 417–426.

[4] Xinyan Fan, Zheng Liu, Jianxun Lian, Wayne Xin Zhao, Xing Xie, and Ji-Rong Wen. 2021. Lighter and better: low-rank decomposed self-attention networks for next-item recommendation. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 1733–1737.

[5] Chen Gao, Yu Zheng, Nian Li, Yinfeng Li, Yingrong Qin, Jinghua Piao, Yuhan Quan, Jianxin Chang, Depeng Jin, Xiangnan He, and Yong Li. 2021. Graph Neural Networks for Recommender Systems: Challenges, Methods, and Directions. (2021). https://arxiv.org/abs/2109.12843

[6] Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *AISTATS 2010*, Vol. 9. 249–256.

[7] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. 2017. DeepFM: A Factorization-Machine based Neural Network for CTR Prediction. In *IJCAI 2017*. 1725–1731.

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

[9] Xiangnan He, Kuan Deng, Xiang Wang, Yan Li, Yong-Dong Zhang, and Meng Wang. 2020. LightGCN: Simplifying and Powering Graph Convolution Network for Recommendation. In *SIGIR 2020*. 639–648.

[10] Xiangnan He, Zhankui He, Jingkuan Song, Zhenguang Liu, Yu-Gang Jiang, and Tat-Seng Chua. 2018. Nais: Neural attentive item similarity model for recommendation. *IEEE Transactions on Knowledge and Data Engineering* 30, 12 (2018), 2354–2366.

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

[12] Cheng-Kang Hsieh, Longqi Yang, Yin Cui, Tsung-Yi Lin, Serge J. Belongie, and Deborah Estrin. 2017. Collaborative Metric Learning. In *WWW 2017*. 193–201.

[13] Tinglin Huang, Yuxiao Dong, Ming Ding, Zhen Yang, Wenzheng Feng, Xinyu Wang, and Jie Tang. 2021. MixGCF: An Improved Training Method for Graph Neural Network-based Recommender Systems. In *KDD 2021*. 665–674.

[14] Jiarui Jin, Jiarui Qin, Yuchen Fang, Kounianhua Du, Weinan Zhang, Yong Yu, Zheng Zhang, and Alexander J. Smola. 2020. An Efficient Neighborhood-based Interaction Model for Recommendation on Heterogeneous Graph. In *KDD*. 75–84.

[15] Wang-Cheng Kang and Julian McAuley. 2018. Self-attentive sequential recommendation. In *ICDM (2018)*. IEEE, 197–206.

[16] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *ICLR 2015*.

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

[18] Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. 2018. Predict then propagate: Graph neural networks meet personalized pagerank. *arXiv preprint arXiv:1810.05997* (2018).

[19] Johannes Klicpera, Stefan Weißenberger, and Stephan Günnemann. 2019. Diffusion Improves Graph Learning. In *NeurIPS 2019*. 13333–13345.

[20] Risi Imre Kondor and John Lafferty. 2002. Diffusion Kernels on Graphs and Other Discrete Structures. In *ICML 2002*. 315–322.

[21] Yehuda Koren. [n. d.]. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *KDD 2008*.

[22] Dawen Liang, Rahul G. Krishnan, Matthew D. Hoffman, and Tony Jebara. 2018. Variational Autoencoders for Collaborative Filtering. In *WWW 2018*. 689–698.

[23] Fan Liu, Zhiyong Cheng, Lei Zhu, Zan Gao, and Liqiang Nie. 2021. Interest-aware Message-Passing GCN for Recommendation. In *WWW 2021*. 1296–1305.

[24] Yao Ma, Xiaorui Liu, Neil Shah, and Jiliang Tang. 2021. Is homophily a necessity for graph neural networks? *arXiv preprint arXiv:2106.06134* (2021).

[25] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. 1999. *The PageRank citation ranking: Bringing order to the web.* Technical Report. Stanford InfoLab.

[26] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2009. BPR: Bayesian Personalized Ranking from Implicit Feedback. In *UAI 2009*. 452–461.

[27] Ruslan Salakhutdinov and Andriy Mnih. 2007. Probabilistic Matrix Factorization. In *NIPS 2007*. 1257–1264.

[28] Yifei Shen, Yongji Wu, Yao Zhang, Caihua Shan, Jun Zhang, Khaled B. Letaief, and Dongsheng Li. 2021. How Powerful is Graph Convolution for Recommendation?. In *CIKM 2021*. 1619–1629.

[29] Jinbo Song, Chao Chang, Fei Sun, Xinbo Song, and Peng Jiang. 2020. NGAT4Rec: Neighbor-Aware Graph Attention Network For Recommendation. (2020). https://arxiv.org/abs/2010.12256

[30] Jianing Sun, Yingxue Zhang, Wei Guo, Huifeng Guo, Ruiming Tang, Xiuqiang He, Chen Ma, and Mark Coates. 2020. Neighbor Interaction Aware Graph Convolution Networks for Recommendation. In *SIGIR 2020*. 1289–1298.

[31] Riku Togashi, Masahiro Kato, Mayu Otani, and Shin'ichi Satoh. 2021. Density-Ratio Based Personalised Ranking from Implicit Feedback. In *WWW*. 3221–3233.

[32] Rianne van den Berg, Thomas N. Kipf, and Max Welling. 2017. Graph Convolutional Matrix Completion. (2017). http://arxiv.org/abs/1706.02263

[33] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2017. Graph attention networks. *arXiv preprint arXiv:1710.10903* (2017).

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

[35] Menghan Wang, Yujie Lin, Guli Lin, Keping Yang, and Xiao-Ming Wu. 2020. M2GRL: A Multi-task Multi-view Graph Representation Learning Framework for Web-scale Recommender Systems. In *KDD 2020*. 2349–2358.

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

[37] Ziyang Wang, Wei Wei, Gao Cong, Xiao-Li Li, Xianling Mao, and Minghui Qiu. 2020. Global Context Enhanced Graph Neural Networks for Session-based Recommendation. In *SIGIR 2020*. 169–178.

[38] Zhen Wang, Zhewei Wei, Yaliang Li, Weirui Kuang, and Bolin Ding. 2022. Graph Neural Networks with Node-wise Architecture. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 1949–1958.

[39] Widder and David Vernon. 1976. *The heat Kernel.* Academic Press 1976.

[40] Le Wu, Xiangnan He, Xiang Wang, Kun Zhang, and Meng Wang. 2021. A Survey on Neural Recommendation: From Collaborative Filtering to Content and Context Enriched Recommendation. (2021). https://arxiv.org/abs/2104.13030

[41] Le Wu, Junwei Li, Peijie Sun, Richang Hong, Yong Ge, and Meng Wang. 2020. Diffnet++: A neural influence and interest diffusion network for social recommendation. *IEEE Transactions on Knowledge and Data Engineering* (2020).

[42] Shiwen Wu, Wentao Zhang, Fei Sun, and Bin Cui. 2020. Graph Neural Networks in Recommender Systems: A Survey. (2020). https://arxiv.org/abs/2011.02260

[43] Yao Wu, Christopher DuBois, Alice X. Zheng, and Martin Ester. 2016. Collaborative Denoising Auto-Encoders for Top-N Recommender Systems. In *WSDM 2016*. 153–162.

[44] Chengfeng Xu, Pengpeng Zhao, Yanchi Liu, Victor S Sheng, Jiajie Xu, Fuzhen Zhuang, Junhua Fang, and Xiaofang Zhou. 2019. Graph Contextualized Self-Attention Network for Session-based Recommendation.. In *IJCAI*, Vol. 19. 3940–3946.

[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 *KDD 2018*. 974–983.

[46] Quanzeng You, Zhengyou Zhang, and Jiebo Luo. 2018. End-to-End Convolutional Semantic Embeddings. In *CVPR 2018*. 5735–5744.

[47] Jialin Zhao, Yuxiao Dong, Ming Ding, Evgeny Kharlamov, and Jie Tang. 2021. Adaptive Diffusion in Graph Neural Networks. In *NeurIPS 2021*, Vol. 34.

[48] Wayne Xin Zhao, Shanlei Mu, Yupeng Hou, Zihan Lin, Yushuo Chen, Xingyu Pan, Kaiyuan Li, Yujie Lu, Hui Wang, Changxin Tian, Yingqian Min, Zhichao Feng, Xinyan Fan, Xu Chen, Pengfei Wang, Wendi Ji, Yaliang Li, Xiaoling Wang, and Ji-Rong Wen. 2021. RecBole: Towards a Unified, Comprehensive and Efficient Framework for Recommendation Algorithms. In *CIKM 2021*. 4653–4664.

A Supplementary

In this supplementary, we present the detailed implementation note of ApeGNN, including the datasets, baselines, evaluation metrics, parameter settings, and the experimental results.

A.1 Datasets

We use public datasets to perform experiments and evaluate the performance of ApeGNN. The detailed descriptions and statistics of these datasets are as follows:

Table 5: The statistics of our experimental datasets on representative models.

Dataset	# Users	# Items	# Interactions	# Density
Ali	106,042	53,591	907,407	0.00016
Amazon	192,403	63,001	1,689,188	0.00014
AMiner	5,340	14,967	163,084	0.00204
Gowalla	29,858	40,981	1,027,370	0.00084
MovieLens	6,040	3,416	999,611	0.04362
Yelp2018	31,668	38,048	1,561,406	0.00130
Epinions	116,260	41,269	188,478	0.00004
ML-1M	6,040	3,706	1,000,209	0.00264
Pinterest	55,187	9,911	1,445,622	0.04468

- **Ali** [13] dataset comes from the Alibaba e-commerce plat-form and interaction record of its users is more than 10 in a user-item bipartite graph.
- **Amazon [13]** is a widely used dataset for recommendation evaluation. We use the Amazon-rec dataset which contains the interaction of the electronics category. Considering there are many preprocessing setting on the Amazon dataset, this paper keeps the same setting as [13].
- **AMiner**[1] collects scientific resource reading behavior from AMiner.org of its users from August to October in 2021. To unify the experimental settings, we remains users and items that each node has at least ten interactions, and splits the interacted records as training, test, and validation set. To evaluate the recommendation performance, for each user, we select 10% of interactions for testing, another 10% of interactions is used for validation and adopting early stopping, and the remaining 80% interactions are used for training.
- **Gowalla**[2] comes from Gowalla website and contains the check-in historical behavior. Wherein, the locations are shared by users as the items and reviews from the user are regarded as interactions.
- **Yelp2018**[3] dataset is obtained from the Yelp challenge in 2018 and includes the interactions between users and lcoal businesses.
- **MovieLens/ML-1M**[4] datasets come from GroupLens Research, whose authors have collected and made available rating data sets from the MovieLens website[5].
- **Epinions/Pinterest** datasets come from RecBole [48] research.

[1]https://www.aminer.cn/data/?nav=openData#AMiner-Paper-Click
[2]http://snap.stanford.edu/data/loc-gowalla.html
[3]https://www.yelp.com/dataset
[4]https://grouplens.org/datasets/movielens/1m/
[5]https://movielens.org

A.2 Baselines

In our experiments, the compared baselines mainly include MF-based models (BPR-MF and NeuMF), first-order models (Mult-VAE and GF-CF), high-order models (NGCF and LightGCN), and attention-based models (NAIS, SASRec, GC-SAN, and LightSANs).

- **BPR-MF** [26] proposes an optimization criterion for personalized ranking by BPR loss which is a matrix factorization-based approach to capture the interactions.
- **NeuMF** [11] exploits collaborative filtering on implicit feedback based on deep neural networks. It leverages a multi-layer perception to model and learns the user-item interaction function with non-linearities.
- **Mult-VAE** [22] is an item-based CF method for implicit feedback by variational autoencoders (VAEs). It adopts the multi-nomial likelihood of data, uses Bayesian inference for parameter estimation, and connects information-theoretic to maximum entropy discrimination.
- **GF-CF** [28] proposes graph filter based CF and proves the special cases via the lens of graph signal processing as well as the importance of smoothness. To compare it with other models fairly, we update its embedding size and test process to keep unity with our test method.
- **NGCF** [36] proposes a message-passing architecture to capture collaborative filtering signal in the first-order and high-order propagation. NGCF stacks embeddings of multiple propagation layers as its final representation for users and items.
- **LightGCN** [9] simplifies the design GCN and adopts the key component—neighborhood aggregation—for collaborative filtering. After obtaining embeddings of all layers, a weighted sum calculation is applied to generate final embedding.
- **NAIS** [10] is an attentive item similarity network which can distinguish each historical item of a user to address the inefficiency issue for item-based CF.
- **SASRec** [15] is a famous sequential-based model to capture long-term interactions. It applies self-attention mechanism to address two issues, which are parsimony of markov chains-based models on sparse data and complexity of RNNs-based models on dense data.
- **GC-SAN** [44] captures rich local dependencies via GNNs and learns graph contextualized representations of items in sequences via attention mechanism.
- **LightSANs** [4] proposes the low-rank decomposed self-attention networks to address the issue of high complexity in self-attention and uncertain position encoding in sequential relations.

A.3 Evaluation Metrics

We apply widely-used metrics to evaluate the top-K performance of the ApeGNN. The detailed description of these metrics are described as follows:

- Recall indicates the coverage of true items as a result of top-K recommendation.

$$Recall@K = \frac{1}{|\mathcal{U}|} \sum_{u_i \in \mathcal{U}} \frac{|T(u_i)| \cap |R(u_i)|}{|T(u_i)|}, s.t. |R(u_i)| = K, \quad (20)$$

where $T(u_i)$ and $R(u_i)$ are the test and recommended item set of user u_i respectively.

- NDCG (Normalized Discounted Cumulative Gain) computes a score which emphasizes higher-ranked true positives. First, the discounted cumulative gain (DCG) is calculated as:

$$DCG@K = \frac{1}{|\mathcal{U}|} \sum_{u_i \in \mathcal{U}} \sum_{k=1}^{K} \frac{2^{(rel_{k,u_i})} - 1}{log_2(2+k)}, \quad (21)$$

where rel_{k,u_i} is 1 if k-th item k is interacted with user u_i, else it is 0. Then the NDCG@K is calculated by Eq. (22):

$$NDCG@K = \frac{DCG@K}{IDCG@K}, \quad (22)$$

where IDCG@K denotes the ideal cumulative gain.

- MRR (Mean Reciprocal Rank) is a measure that can return a list of the correctly-recommended item to test users.

$$MRR@K = \frac{1}{T} \sum_{i \in R_u} \frac{1}{Rank(i)}, \quad (23)$$

where T is the user's number in test set.

- HR (Hit Ratio) represents the proportion of the total number of test sets in the top-K list of each user to all test sets.

$$Hit\ Ratio@K = \frac{\sum_{i \in U} R(i)}{\sum_{i \in U} T(i)}, \quad (24)$$

where $U, T(i), R(i)$ are user set, recommended set of user i, and test set of user i, respectively.

A.4 Implementation note

Running Environment. We conduct all experiments on Ubuntu 18.04.2 LTS server with Intel(R) Xeon(R) CPU E5-2680 v4 @ 2.40GHz, 252G RAM and 8 NVIDIA GeForce RTX 2080TI-11GB. We implement ApeGNN with Python 3.7.6 and PyTorch 1.7.0.

Hyper-parameter Settings We implement our ApeGNN in PyTorch. For all models, we set batch size as 2048. We optimize ApeGNN with a learning rate at 0.001 and Adam optimizer [16]. Meanwhile, we set the propagation layer from 1 to 4, the embedding dimension in range of $\{64, 128, 256, 512\}$, and the coefficient λ of L_2 normalization in range of $\{10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}\}$. In terms of hyper-parameters, to find the best settings, we apply a grid search. For Ali, Amazon, AMiner, Gowalla, MovieLens, and Yelp2018, the L_2 coefficient λ are 10^{-3}, 10^{-2}, 10^{-3}, 10^{-3}, 10^{-2}, and 10^{-3}, respectively. Besides, to initialize the model parameters, we use the Xavier initializer [6]. Moreover, if Recall@20 does not increase for 10 consecutive epochs on the validation dataset, we apply an early stopping strategy to stop training.

Training Algorithm. We show the training algorithm of ApeGNN in Algorithm 1.

Algorithm 1: The training algorithm of the ApeGNN.

Input: $\mathcal{G}$: user-item bipartite graph,
$\mathcal{U}$: user set, $\mathcal{V}$: item set, L: number of layer.

Initialize $\mathbf{h}_{u_i}^{(0)}, \mathbf{h}_{v_j}^{(0)}, \forall u_i \in \mathcal{U}, \forall v_j \in \mathcal{V}$
$iter \leftarrow 0$
while $\mathcal{L}$ is not converged **do**
  for $l \leftarrow 0$ **to** L **do**
    $\mathbf{h}_{u_i}^{(l)}$ = Aggregate and Propagate by Eq. (7) and Eq. (16), $\forall u_i \in \mathcal{U}$
    $\mathbf{h}_{v_j}^{(l)}$ = Aggregate and Propagate by Eq. (8) and Eq. (16), $\forall v_j \in \mathcal{V}$
  end for
  $\mathbf{h}_{u_i}^{(0)} \leftarrow h_{u_i}^{*}$ = Pooling by Eq.(18), $\forall u_i \in \mathcal{U}$
  $\mathbf{h}_{v_j}^{(0)} \leftarrow h_{v_j}^{*}$ = Pooling by Eq.(18), $\forall v_j \in \mathcal{V}$
  $\hat{\mathcal{R}} \leftarrow \hat{r}_{(u_i,v_j)} = h_{u_i}^{*} \odot h_{v_j}^{*}, \forall u_i \in \mathcal{U}, \forall v_j \in \mathcal{V}$
  Update $\mathcal{L}$ with BPR Loss by Eq. (19)
  $iter \leftarrow iter + 1$
end while

Output: $\hat{\mathcal{R}} \in \mathbb{R}^{m \times n}$: preferences for users on candidated items.

Additional Results. We show effectiveness of layer number on the other three datasets (Gowalla, MovieLens and Yelp2018) in Table 6.

Table 6: Results on top-20 recommendation between LightGCN and ApeGNN on other datasets at different layers. Underline and bold denote the best results on LightGCN and ApeGNN, respectively.

Dataset		Gowalla		Yelp2018		MovieLens	
#Layers	Method	Recall	NDCG	Recall	NDCG	Recall	NDCG
1 Layer	LightGCN	16.38	14.02	5.75	4.66	9.30	11.25
	ApeGNN	17.86	15.04	6.55	5.39	**9.73**	**11.91**
2 Layers	LightGCN	16.99	14.60	6.12	5.00	9.41	11.39
	ApeGNN	18.16	15.24	6.72	5.53	9.67	11.87
3 Layers	LightGCN	17.58	15.02	6.34	5.15	9.41	11.36
	ApeGNN	18.28	**15.36**	**6.75**	**5.56**	9.65	11.86
4 Layers	LightGCN	17.75	15.22	6.61	5.39	9.31	11.34
	ApeGNN	**18.32**	15.35	6.67	5.46	9.62	11.83