World Wide Web (2023) 26:2835–2854
https://doi.org/10.1007/s11280-023-01157-z

Quaternion-based graph convolution network for recommendation

Yaxing Fang, Pengpeng Zhao, Guanfeng Liu, Yanchi Liu, Victor S. Sheng, Lei Zhao, et al. ***[full author details at the end of the article]***

Received: 7 August 2022 / Revised: 16 September 2022 / Accepted: 22 February 2023 /
Published online: 15 May 2023

Abstract
Graph Convolution Network (GCN) has been widely applied in recommender systems for its representation learning capability on user and item embeddings. However, GCN is vulnerable to noisy and incomplete graphs, which are common in real world, due to its recursive message propagation mechanism. In the literature, some work proposes to remove the feature transformation during message propagation, but making it unable to effectively capture the graph structural features. Moreover, they model users and items in the Euclidean space, which has been demonstrated to have high distortion when modeling complex graphs, further degrading the capability to capture the graph structural features and leading to sub-optimal performance. To this end, in this paper, we propose a simple yet effective Quaternion-based Graph Convolution Network (QGCN) recommendation model. In the proposed model, we utilize the hyper-complex Quaternion space to learn user and item representations and feature transformation to improve both performance and robustness. Specifically, we first embed all users and items into the Quaternion space. Then, we introduce the quaternion embedding propagation layers with quaternion feature transformation to perform message propagation. Finally, we combine the embeddings generated at each layer with the mean pooling strategy to obtain the final embeddings for recommendation. Extensive experiments on three public benchmark datasets demonstrate that our proposed QGCN model outperforms baseline methods by a large margin.

Keywords Recommender systems · Collaborative filtering · Graph neural network · Quaternion embedding

1 Introduction

Recommender systems have been widely used for alleviating information overload in real-world applications, such as social media [16], news [42], videos [39], E-commerce [15] and Point of Interest (POI) applications [30]. It aims to estimate whether a user will show a prefer-

✉ Pengpeng Zhao
ppzhao@suda.edu.cn

Extended author information available on the last page of the article

ence for an item, based on the user's historical interactions. Among existing recommendation methods, Collaborative Filtering (CF) based models [8, 14, 34, 36] have shown great performance in user and item representation learning. For example, Matrix Factorization [13] represents users and items with embedding vectors and models the user-item interactions with the inner product. Neural collaborative filtering [8] utilizes nonlinear neural networks with multiple hidden layers to capture the user-item interactions for better user and item representations.

Recently, GCN-based recommendation models have surged to learn better user and item representations in the user-item bipartite graph. The typical flow can be summarized as follows: 1) Initialize user and item representations by embedding them into the latent space; 2) Use an aggregation function over neighbors of each node to update its representation iteratively; 3) Readout the final representation of each node by combining or concatenating. The paradigm of GCN iterative aggregating feature information from local graph neighbors has been proved to be an efficient way to distill additional information from graph structure and thus improve user and item representation learning. For example, GC-MC [1] explores the first-order connectivity between users and items by utilizing only one convolution layer in the user-item bipartite graph. NGCF [32] leverages the message-passing mechanism to obtain high-order connectivity and collaborative signal between users and items. LightGCN [9] simplifies the NGCF [32] model, removing the components of feature transformation and nonlinear activation, leading to improvement in training efficiency and generation ability.

Despite effectiveness, GCN is still vulnerable to noisy and incomplete graphs, which are common in real-world scenarios, due to its recursive message propagation mechanism [3, 4, 41]. However, some latest GCN-based recommendation models (e.g. LightGCN [9]) propose to remove the feature transformation during message propagation, but making it unable to effectively capture the graph structural features and become more sensitive to noisy or missing information. Moreover, they model users and items with real-value embeddings, which has been demonstrated to have high distortion when modeling complex graphs [2, 18], further degrading the capability to capture the graph structural features and leading to sub-optimal performance.

Can we move beyond the Euclidean space to learn better user and item representations and feature transformation, capture the graph structural features more effectively, and thus improve both recommendation performance and model robustness? Quaternion space - a hyper-complex vector space, where each quaternion is a hyper-complex number consisting of one real and three imaginary components, has shown great performance in representation learning [19, 20, 38]. Hamilton product, the multiplication of quaternions, enhances the inter-latent interactions between real and imaginary components of two quaternions. Any slight change in the input quaternion results in an entirely different output, leading to highly expressive computations, thus the intricate relations are captured more powerfully [25]. Figure 1 shows the difference between real-value transformation and quaternion transformation. There has been significant success of quaternion-based methods in various fields. For example, [5] builds deep quaternion networks in the Quaternion space for classification tasks. Zhu et al. [43] proposes quaternion-based convolutional neural network (CNN) for image classification and denoising tasks. Parcollet et al. [24] applies the Quaternion space into recurrent neural network (RNN) and long-short term memory neural network (LSTM) for automatic speech recognition. Parcollet et al. [23] integrates multiple feature views in quaternion-valued CNN to be used for sequence-to-sequence mapping with the CTC model. [22] investigates quaternion-based CNN and RNN for speech recognition. [28] proposes quaternion-based attention models and Transformer for NLP tasks.

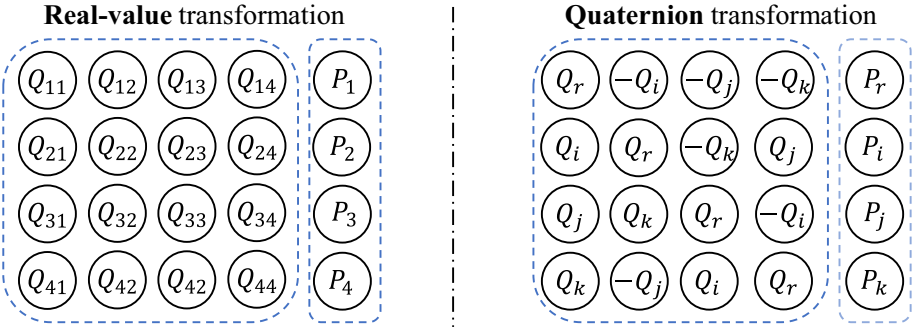

Figure 1 Comparison between real-value transformation and quaternion transformation

Moreover, there has been some work introducing the Quaternion space into graph representation learning to obtain more expressive graph-level representations [19, 20, 38]. For example, [19] generalizes graph neural networks in the Quaternion space for graph classification, node classification, and text classification. Nguyen et al. [20], Zhang et al. [38] introduce more expressive quaternion representations to model entities and relations for knowledge graph embeddings for knowledge graph completion. Despite Quaternion space being introduced into various fields for various tasks and achieving remarkable performance improvement, there is almost no exploration of the Quaternion space in GCN-based recommendation scenarios. Some challenges during this process remain to be explored. The most crucial one is that: The model should not be designed to be very complex or redundant to better validate the effectiveness of the Quaternion space and for more intuitive comparison. In other words, how to introduce the Quaternion space while keeping the model as simple as possible remains to be considered.

To this end, in this paper, we propose a simple yet effective **Q**uaternion-based **G**raph **C**onvolution **N**etwork (QGCN) recommendation model, which improves both performance and robustness. Specifically, we first embed all users and items into the Quaternion space with quaternion embeddings. Then, we introduce the quaternion embedding propagation layers with quaternion feature transformation to perform message propagation for aggregating more useful information. Finally, we combine the embeddings generated at each layer with the mean pooling strategy to obtain the final embeddings for recommendation. The quaternion feature transformation enhances the inter-latent interactions between real and imaginary components, enabling it to capture the graph structural features more effectively, distinguish the contribution of different nodes during message propagation, and thus improve both performance and robustness. Extensive experiments are conducted on three public benchmark datasets to validate the effectiveness of our proposed QGCN model. Results show that QGCN outperforms the state-of-the-art methods by a large margin, which indicates that it can better learn user and item representations. Besides, with further robustness analysis, we find that the performance of our QGCN model remains steady in various noisy or incomplete graphs, while that of compared state-of-the-art methods declines dramatically. This indicates that our model is more robust and can effectively capture the graph structural features.

We summarize the contributions of this work as follows:

- To the best of our knowledge, we are the first to introduce the Quaternion space into GCN-based recommendation models.

- A QGCN model is proposed to model users and items in the Quaternion space and propagate them with quaternion feature transformation, which significantly enhances both recommendation performance and model robustness.
- We conduct extensive experiments on three public benchmark datasets, and experimental results demonstrate the effectiveness of our QGCN model. Results of robustness analysis verify the effectiveness of the quaternion feature transformation in capturing the graph structural features.

2 Preliminaries

In this section, we cover some necessary background on quaternion before delving into the architecture of our proposed model.

2.1 Quaternion

A quaternion $Q \in \mathbb{H}$ is a hyper-complex number consisting of one real part and three imaginary parts defined as:

$$Q = Q_r + Q_i\mathbf{i} + Q_j\mathbf{j} + Q_k\mathbf{k}, \tag{1}$$

where $Q_r, Q_i, Q_j, Q_k \in \mathbb{R}$, and $\mathbf{i}, \mathbf{j}, \mathbf{k}$ are imaginary units, satisfying the following rule:

$$\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = \mathbf{ijk} = -1. \tag{2}$$

A n-dimensional vector form of quaternion $\boldsymbol{Q} \in \mathbb{H}^n$ is defined as:

$$\boldsymbol{Q} = \boldsymbol{Q_r} + \boldsymbol{Q_i}\mathbf{i} + \boldsymbol{Q_j}\mathbf{j} + \boldsymbol{Q_k}\mathbf{k}, \tag{3}$$

where $\boldsymbol{Q_r}, \boldsymbol{Q_i}, \boldsymbol{Q_j}, \boldsymbol{Q_k} \in \mathbb{R}^n$, each coefficient of the real unit or the imaginary units is a n-dimensional vector.

2.2 Quaternion addition

The addition of two quaternions Q and P is defined as:

$$Q + P = (Q_r + P_r) + (Q_i + P_i)\mathbf{i} + (Q_j + P_j)\mathbf{j} + (Q_k + P_k)\mathbf{k}, \tag{4}$$

2.3 Quaternion inner product

The inner product of two quaternions Q and P is defined as:

$$Q \cdot P = Q_r \cdot P_r + Q_i \cdot P_i + Q_j \cdot P_j + Q_k \cdot P_k. \tag{5}$$

2.4 Hamilton product

The quaternion product of two quaternions Q and P is defined as:

$$\begin{aligned} Q \otimes P = &(Q_r P_r - Q_i P_i - Q_j P_j - Q_k P_k) \\ &+(Q_i P_r + Q_r P_i - Q_k P_j + Q_j P_k)\mathbf{i} \\ &+(Q_j P_r + Q_k P_i + Q_r P_j - Q_i P_k)\mathbf{j} \\ &+(Q_k P_r - Q_j P_i + Q_i P_j + Q_r P_k)\mathbf{k}. \end{aligned} \tag{6}$$

We further simplify the result of Hamilton product above into matrix form as follows:

$$\begin{bmatrix} 1 \\ \mathbf{i} \\ \mathbf{j} \\ \mathbf{k} \end{bmatrix}^{\mathrm{T}} \begin{bmatrix} Q_r & -Q_i & -Q_j & -Q_k \\ Q_i & Q_r & -Q_k & Q_j \\ Q_j & Q_k & Q_r & -Q_i \\ Q_k & -Q_j & Q_i & Q_r \end{bmatrix} \begin{bmatrix} P_r \\ P_i \\ P_j \\ P_k \end{bmatrix}. \tag{7}$$

3 Methodology

In this section, we present our proposed QGCN model. As illustrated in Figure 2, the model contains three main components: Quaternion Embedding Layer, Quaternion Embedding Propagation Layers, and Prediction Layer.

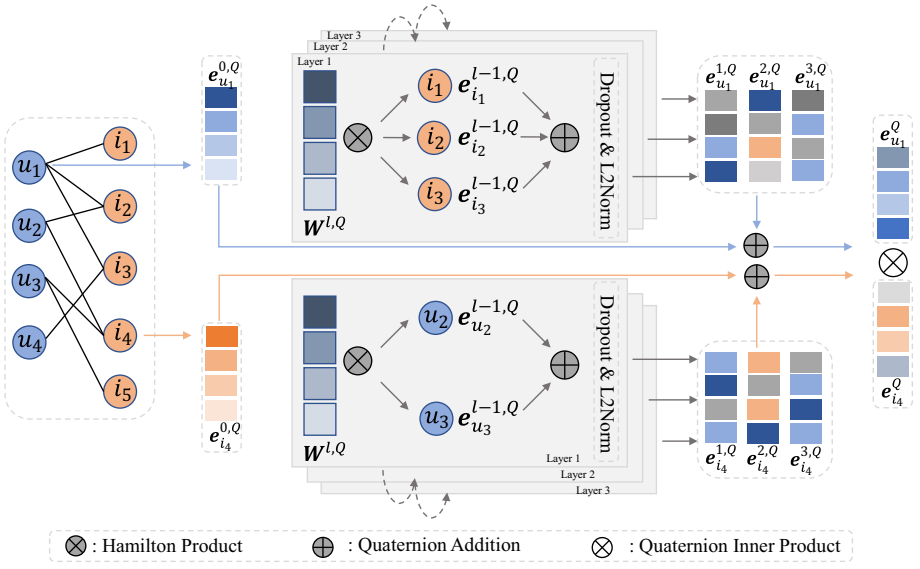

Figure 2 The architecture of our proposed QGCN model which is formed by Quaternion Embedding Layer, Quaternion Embedding Propagation Layers and Prediction Layer

3.1 Quaternion embedding layer

Firstly, we embed all the users and items into the Quaternion space. For each user $u \in \mathcal{U}$, we represent it with a quaternion ID embedding $\mathbf{e}_u^{0,Q} = \mathbf{e}_{u,r}^0 + \mathbf{e}_{u,i}^0\mathbf{i} + \mathbf{e}_{u,j}^0\mathbf{j} + \mathbf{e}_{u,k}^0\mathbf{k} \in \mathbb{H}^d$, where d represents the dimension of quaternion. And the same for item quaternion ID embeddings.

3.2 Quaternion embedding propagation layers

3.2.1 Quaternion embedding propagation

Next, we perform message propagation within the Quaternion Embedding Propagation Layers with quaternion feature transformation. As mentioned above, we argue that removing the feature transformation during message propagation makes it unable to effectively capture the graph structural features and become more sensitive to noisy or missing information, further degrading the model performance. So in this part, we introduce the feature transformation in the Quaternion space at each layer for message propagation to aggregate more useful information. In order to prove our quaternion feature transformation to be valid more intuitively, we adopt the simple message propagation procedure like the vanilla GCN [12] without the nonlinear activation function, only involving the user and item embeddings and the quaternion transformation matrices. We generate the quaternion transformation matrix at layer l as follows:

$$\mathbf{W}^{l,Q} = \mathbf{W}_r^l + \mathbf{W}_i^l\mathbf{i} + \mathbf{W}_j^l\mathbf{j} + \mathbf{W}_k^l\mathbf{k}, \tag{8}$$

where $\mathbf{W}_r^l, \mathbf{W}_i^l, \mathbf{W}_j^l, \mathbf{W}_k^l \in \mathbb{R}^{d\times d}$.

Thus, our quaternion embedding propagation rule in QGCN is defined as:

$$\begin{aligned}\mathbf{e}_u^{l,Q} &= \sum_{i\in\mathcal{N}_u} \frac{1}{\sqrt{|\mathcal{N}_u||\mathcal{N}_i|}}\mathbf{W}^{l,Q} \otimes \mathbf{e}_i^{l-1,Q},\\ \mathbf{e}_i^{l,Q} &= \sum_{u\in\mathcal{N}_i} \frac{1}{\sqrt{|\mathcal{N}_i||\mathcal{N}_u|}}\mathbf{W}^{l,Q} \otimes \mathbf{e}_u^{l-1,Q},\end{aligned} \tag{9}$$

where $\mathbf{e}_u^{l,Q}$ and $\mathbf{e}_i^{l,Q}$ respectively represent user u's quaternion embedding and item i's quaternion embedding after l layers propagation; $1/\sqrt{|\mathcal{N}_u||\mathcal{N}_i|}$ is the symmetric normalization term following the vanilla GCN [12], designed to avoid the scale of embeddings increasing with graph convolution operations, where $\mathcal{N}_u$ and $\mathcal{N}_i$ respectively denote the user u's interacted items and the item i's interacted users; $\mathbf{W}^{l,Q} \in \mathbb{H}^{d\times d}$ is the quaternion feature transformation matrix at layer l; $\otimes$ denotes Hamilton product.

To facilitate the implementation of the quaternion embedding propagation, we derive the Hamilton product $\otimes$ between $\mathbf{W}^{l,Q}$ and $\mathbf{e}_u^{l-1,Q}$ in Eq. (9) as follows (*c.f.* (7)):

$$\begin{bmatrix}1\\ \mathbf{i}\\ \mathbf{j}\\ \mathbf{k}\end{bmatrix}^{\mathrm{T}} \begin{bmatrix}\mathbf{W}_r^l & -\mathbf{W}_i^l & -\mathbf{W}_j^l & -\mathbf{W}_k^l\\ \mathbf{W}_i^l & \mathbf{W}_r^l & -\mathbf{W}_k^l & \mathbf{W}_j^l\\ \mathbf{W}_j^l & \mathbf{W}_k^l & \mathbf{W}_r^l & -\mathbf{W}_i^l\\ \mathbf{W}_k^l & -\mathbf{W}_j^l & \mathbf{W}_i^l & \mathbf{W}_r^l\end{bmatrix} \begin{bmatrix}\mathbf{e}_{u,r}^{l-1}\\ \mathbf{e}_{u,i}^{l-1}\\ \mathbf{e}_{u,j}^{l-1}\\ \mathbf{e}_{u,k}^{l-1}\end{bmatrix}. \tag{10}$$

The result of Hamilton product $\otimes$ between $\mathbf{W}^{l,Q}$ and $\mathbf{e}_i^{l-1,Q}$ can be derived similarly.

3.2.2 Dropout and L2Norm

Dropout drops the units of the neural networks with a certain probability during the training process, which proves to be an effective way to prevent neural networks from overfitting [10, 27]. Motivated by the previous work of introducing dropout into graph convolutional network [1] and GCN-based recommendation models [32], we apply dropout to the user and item embeddings at each layer l with a certain dropout rate p, which is one of the critical hyper-parameters to be tuned. Then, we perform L2 Normalization function on them for training speed and stability. We summarize the dropout and L2 normalization as follows:

$$\begin{aligned} \mathbf{e}_u^{l,Q} &= L2Norm\left(Dropout(\mathbf{e}_u^{l,Q})\right), \\ \mathbf{e}_i^{l,Q} &= L2Norm\left(Dropout(\mathbf{e}_i^{l,Q})\right). \end{aligned} \tag{11}$$

3.3 Prediction layer

After the above L layers' quaternion embedding propagation, dropout and L2 normalization, we obtain $L+1$ representations for each user u and item i, including the user embedding initialized at quaternion embedding layer, $\mathbf{e}_u^{0,Q}$ and user representations generated at each layer during propagation, $\{\mathbf{e}_u^{1,Q}, \mathbf{e}_u^{2,Q}, \ldots, \mathbf{e}_u^{L,Q}\}$. And the same for item i, we obtain $L+1$ item representations which consist of $\{\mathbf{e}_i^{0,Q}, \{\mathbf{e}_i^{1,Q}, \mathbf{e}_i^{2,Q}, \ldots, \mathbf{e}_i^{L,Q}\}\}$. Since the output of different layers expresses different connections, utilizing the representations of all layers seems like an effective method for GCN-based models. Readout function is the method to obtain the final node representation, e.g. Max, Sum, Concat, Mean pooling, which are the most primitive and simple pooling methods.

We concatenate the real and imaginary components of the node embeddings and apply the pooling methods as follows:

$$\mathbf{e}_u^l = Concat\{\mathbf{e}_{u,r}^{l,Q}, \mathbf{e}_{u,i}^{l,Q}\mathbf{e}_{u,j}^{l,Q}, \mathbf{e}_{u,k}^{l,Q}\}, \tag{12}$$

$$\mathbf{e}_u^* = Readout\{\mathbf{e}_u^l\}_{l=1}^L, \tag{13}$$

where *Readout* is the readout function (i.e. Max, Sum, Concat, Mean pooling) applied on the node embeddings generated at each layer. We further conduct experiments and investigate the influence of the readout function applied to our model in the ablation study part.

After generating the final user and item embeddings, we predict by the inner product of user u and item i:

$$\hat{y_{ui}} = \mathbf{e}_u^{*\mathrm{T}}\mathbf{e}_i^*. \tag{14}$$

3.4 Optimization

We adopt *Bayesian Personalized Ranking* (BPR) loss [26], which encourages the observed interactions to achieve higher scores than the unobserved ones. The objective function for our QGCN model is as follows:

$$Loss = \sum_{u=1}^{M} \sum_{i \in \mathcal{N}_u} \sum_{j \notin \mathcal{N}_u} -\ln \sigma(\hat{y}_{ui} - \hat{y}_{uj}) + \lambda \|\mathbf{\Theta}\|_2^2, \tag{15}$$

where σ is the sigmoid function; λ represents the regularization weight, which is L_2 regularization to prevent overfitting; $\mathbf{\Theta} = \left\{\{\mathbf{e}_u^{0,\mathcal{Q}}\}_{u\in\mathcal{U}}, \{\mathbf{e}_i^{0,\mathcal{Q}}\}_{i\in\mathcal{I}}, \{\mathbf{W}^{l,\mathcal{Q}}\}_{l\in[1,L]}\right\}$ denotes all trainable parameters of QGCN. The mini-batch Adam [11] is adopted to optimize the prediction model and update the model parameters. In particular, for a batch of randomly sampled triples, their representations can be obtained by the propagation rules, and then the model parameters are updated by using the gradients of the loss function.

3.5 Complexity analysis

Specifically, $\mathbf{R}$, $|V|$ and $|E|$, L and d respectively represent the user-item interaction matrix, the number of nodes and edges, the number of graph convolution layers, and the quaternion dimension.

3.5.1 Time complexity

The time complexity of our model is mainly in the following three parts, adjacency matrix, graph convolution, and BPR loss. For the adjacency matrix, the time complexity is $\mathcal{O}(|E|)$ that we set each element $\mathbf{R}_{ui} = 1$ in user-item interaction matrix $\mathbf{R}$ if user u has interacted with item i. For the graph convolution, the quaternion embedding propagation has computation complexity $\mathcal{O}(L|E|d^2)$. For the BPR loss, the time complexity is $\mathcal{O}(|E|d)$. Therefore, the overall time complexity of our model is $\mathcal{O}(|E| + L|E|d^2 + |E|d)$. As we can see, the layer-wise propagation rule is the main operation. By deriving the Hamilton product in (9) with the real-value operation (10), we can conclude that the time complexity of the quaternion propagation is equal to the normal matrix multiplication in the basic GCN propagation rule leveraged by previous GCN-based recommendation models.

3.5.2 Space complexity

The space complexity of our model is mainly in the user and item embeddings and the quaternion transformation matrix at each layer. Therefore, the overall space complexity of our model is $\mathcal{O}(|V|d + Ld^2)$, which is equal to the space complexity of previous GCN-based recommendation models.

4 Experiments

In this section, we conduct experiments to answer the following research questions:

- **RQ1** How does our proposed QGCN model perform compared with the state-of-the-art baselines?
- **RQ2** How can QGCN alleviate the problem of noisy or incomplete graphs?
- **RQ3** What is the influence of readout function, quaternion embedding and quaternion weight matrices on the model performance?
- **RQ4** How do the key hyper-parameters, such as dropout rate and regularization affect the effectiveness of QGCN?

Table 1 Statistics of the experimented data

Dataset	#Users	#Items	#Interactions	#Density
Yelp2018	31668	38048	1561406	0.00130
Amazon-Book	52643	91599	2984108	0.00062
Kindle-Store	68223	61934	982618	0.00023

4.1 Datasets

To evaluate the effectiveness of QGCN, we conduct experiments on three benchmark datasets: Yelp2018 [9], Amazon-Book [9], and Amazon-Kindle-Store [17], which are publicly available. The first dataset is the 2018 edition Yelp[1] released by the Yelp challenge. The last two datasets are two widely used datasets for product recommendation from Amazon review.[2]

Following the general dataset settings in previous recommendation methods [9, 17, 32], we filter users and items with few interactions to ensure the quality of the datasets. Specifically, for all the datasets, we use the 10-core settings, which ensure that each user and item have at least 10 interactions. The detailed statistics of the three datasets are shown in Table 1.

We randomly split each dataset into training, validation, and testing set with a ratio of 80:10:10 for each user. For each observed user-item interaction, we treat it as a positive instance. Then, we randomly sample one negative item that the user did not consume before as a negative instance to pair the positive instance.

4.2 Experimental settings

4.2.1 Evaluation metrics

To evaluate the effectiveness of our model on top-K recommendation, we take two evaluation metrics widely used in previous work: Recall@K and NDCG@K. Here, we set $K = 20$ by default, and the average results for all users in the testing set are reported. The specific definition is as follows:

- Recall@K describes the percentage of user-item rating records included in the final recommendation list. We denote the recommendation list for a user as R_K, and the corresponding testing set as T. Then, the specific definition of Recall@K is as follows:

$$\text{Recall@K} = |T \cap R_K|/|T|. \tag{16}$$

- NDCG@K i.e. Normalized Discounted Cumulative Gain measures the quality of ranking, which emphasizes more on the relevance of the items on the top of the recommendation list. We denote the relevance of the i-th item in the recommendation list as r_i, and the set of relevant items as R. Then, the specific definition of NDCG@K is:

$$\text{NDCG@K} = \sum_{i=1}^{K} \frac{r_i}{\log_2(i+1)} \Big/ \sum_{i=1}^{\text{R}} \frac{1}{\log_2(i+1)}, \tag{17}$$

[1] https://www.yelp.com/dataset

[2] https://jmcauley.ucsd.edu/data/amazon/

Table 2 Overall performance comparison over three datasets

Dataset	Yelp2018		Amazon-Book		Kindle-Store	
Metric	Recall@20	NDCG@20	Recall@20	NDCG@20	Recall@20	NDCG@20
NeuMF	0.0451	0.0363	0.0258	0.0200	0.0496	0.0206
HOP-Rec	0.0517	0.0428	0.0309	0.0232	0.0796	0.0458
GC-MC	0.0462	0.0379	0.0288	0.0224	0.0793	0.0455
NGCF	0.0579	0.0477	0.0344	0.0263	0.0825	0.0509
LightGCN	0.0649	0.0530	0.0411	0.0315	0.1040	0.0639
QGCN	**0.0668**	**0.0547**	**0.0489**	**0.0376**	**0.1250**	**0.0788**
%Improv.	2.93%	3.21%	18.98%	19.37%	20.19%	23.32%

4.2.2 Baselines

To demonstrate the effectiveness of our proposed QGCN model, we compare QGCN with the following competitive baseline methods:

- **NeuMF** [8] is a state-of-the-art neural collaborative filtering model, captures the nonlinear interactions between user and item embeddings with multiple hidden layers.
- **HOP-Rec** [37] is a state-of-the-art graph-based model, exploits the high-order connectivity between users and items by performing random walks to augment a user's interactions.
- **GC-MC** [1] explores the first-order connectivity between users and items by utilizing only one convolution layer over the user-item bipartite graph.
- **NGCF** [32] leverages the message-passing mechanism to obtain high-order connectivity and collaborative signal in the user-item integration graph.
- **LightGCN** [9] removes two components, feature transformation and nonlinear activation in NGCF, leading to improvement on training efficiency and generation ability.

4.2.3 Parameter settings

We implement our QGCN model in PyTorch,[3] optimize QGCN with Adam [11] with the default learning rate of 0.0001, and set batch size as 2048 for speed. We apply a grid search for the only two hyper-parameters: the dropout rate is tuned among {0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8} and the coefficient of L_2 normalization in Equation (15) is searched in $\{1e^{-6}, 1e^{-5}, \ldots, 1e^{-2}\}$. The embedding parameters are initialized with the Xavier method [6].

4.3 Performance comparison (RQ1)

Table 2 shows the performance with competing methods. The best results are highlighted in bold. We summarize the main observations as follows:

- NeuMF, a state-of-the-art neural collaborative filtering model, performs relatively poorly since it captures the connectivity between user and item embeddings in the embedding learning process rather than leveraging the high-order user-item interactions.

[3] https://pytorch.org

- Compared with NeuMF, GC-MC utilizes one convolution layer to explore the first-order connectivity between users and items and improve the performance, demonstrating the influence of first-order neighbors for representation learning.
- HOP-Rec exploits the high-order connectivity between users and items by performing random walks to augment a user's interactions, resulting in better performance than GC-MC. NGCF performs much better over the above baselines. It leverages the message-passing mechanism to obtain high-order connectivity and collaborative signal in the user-item integration graph. LightGCN simplifies the NGCF [32] model, removing the components of feature transformation and nonlinear activation, leading to improvement in training efficiency and generation ability.
- QGCN outperforms all the baselines by a large margin over all the datasets. In particular, compared with the strongest baseline, i.e LightGCN, QGCN gains on average 14.03% improvement *w.r.t.* Recall@20 and 15.30% improvement *w.r.t.* NDCG@20 over all the datasets. In collaborative filtering, the learned user/item representation getting as close as possible to its actual representation is necessary for better recommendation performance. The significant improvements reveal that QGCN can better capture high-order user-item connectivity and learn better user and item embeddings.
- Specifically, our QGCN model gains huge and about 20%, relative performance improvement. We ascribe this to the characteristics of the datasets, sparsity. As the sparsity of the dataset decreases, the quaternion feature transformation could highlight its contribution to distilling sufficient information from the sparse user-item interaction graphs and further lead to more significant performance improvement. Aside from the effect of sparsity, we conclude that the user-item interaction matrix size might matter as well. The size of the Amazon-Book and Kindle-Store user-item interaction matrix is about four times that of Yelp2018. Moreover, the observations mentioned above that our QGCN model gaining huge relative performance improvement on sparse and huge graphs is of great significance to the practical applications and real recommendation scenarios since real-world graphs for recommendation are often extremely sparse.
- For more intuitive comparisons, We further plot the training curves of training loss and testing recall per 10 epochs on Kindle-Store and Amazon-Book with optimal settings on both LightGCN and our QGCN model in Figure 3, where results on Yelp2018 show the same trend and are omitted for space. Our QGCN model obtains relatively lower training loss during the whole training process than that in LightGCN, which indicates our QGCN model can better fit the training data and further obtains better testing results, which demonstrates our model's stronger generalization capability.

4.4 Robustness analysis (RQ2)

4.4.1 Random Edges Injection

To investigate the robustness of our QGCN model to noisy graphs, we conduct simulated experiments to explore the influence of random injection of edges. Specifically, we randomly connect the unobserved edges in the user-item interaction graph $\mathbf{R}$ as noisy edges to construct a noisy graph for the training process. The noise ratio is set in {5%, 10%, 15%, 20%, 25%}. By the way, the compared LightGCN model and our QGCN model are trained with the same constructed noisy graph for a fair comparison. And we evaluate with the original graph (i.e. 0% edges injection). We further plot Recall@20 and relative drop compared with their

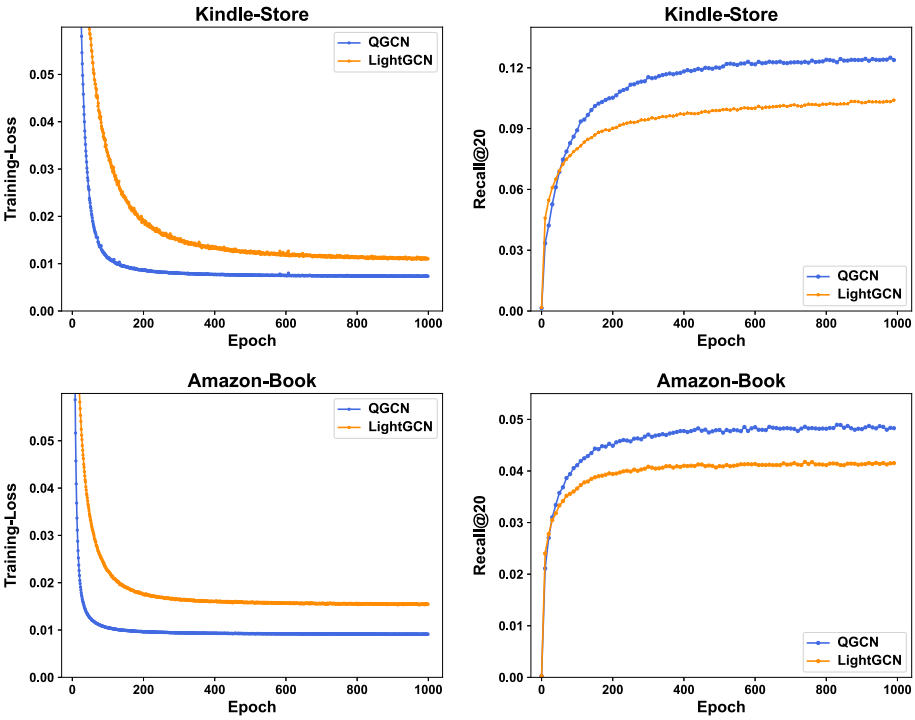

Figure 3 Training curves of QGCN and LightGCN, which are evaluated by training loss and testing recall per 10 epochs on Kindle-Store and Amazon-Book(results on Yelp2018 show the same trend which are omitted for space)

original performance of both LightGCN and our QGCN model on Kindle-Store and Yelp2018 in Figure 4.

We observe that QGCN consistently outperforms LightGCN by a large margin under different ratios of random edges injection on both Kindle-Store and Yelp2018. Along with the increase of the noise ratio, the performance of LightGCN decreases accordingly, while that of our QGCN model remains almost unchanged. For example, Recall@20 of LightGCN

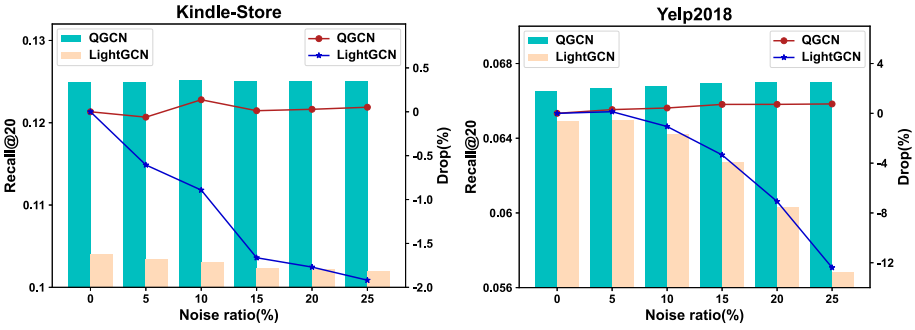

Figure 4 Effect of random edges injection. The bar represents Recall@20, while the line represents the relative performance change compared to the original result

in the noisy graph with 25% noise ratio of noise injection on Yelp2018 is 0.0568, dropping 12.48% (i.e. -12.48%) compared to the original performance, 0.0649. In contrast to the large drop percent of LightGCN, the performance of our QGCN model under 25% noise ratio even rises by 0.75% (i.e. +0.75%) compared to that under 0% noise ratio. The sharp decline of the relative drop of Recall@20 of LightGCN along with the increase of noise ratio reveals that LightGCN is extremely sensitive to noise, which is consistent with our argument mentioned before. Compared with the steep decline curve of Recall@20 of LightGCN, the relative performance change curve of our QGCN model is more steady, which demonstrates the robustness of our QGCN model to noisy graphs.

4.4.2 Random edges discard

In addition to the characteristic of real-world user-item graphs containing a lot of noise, they are often incomplete as well. Thus, besides the simulated experiments on exploring the influence of random injection of edges, we also conduct experiments to explore the influence of the random discard of edges. Similarly, we construct a corrupted graph by randomly disconnect the existing edges in the user-item interaction graph $\mathbf{R}$ with a drop ratio ranging in {5%, 10%, 15%, 20%, 25%}. We then train the compared LightGCN model and our QGCN model with the corrupted graph and evaluate with the original graph (i.e. 0% edges discard). The details of Recall@20 and relative drop are shown in Figure 5.

We have similar observations from Figure 5. Specifically, QGCN consistently outperforms LightGCN by a large margin *w.r.t* different ratios of random edges discard on both Kindle-Store and Yelp2018. The steep performance decline curve of LightGCN is in sharp contrast to the steady curve of QGCN, demonstrating the robustness of our QGCN model to corrupted graphs.

The simulated experiments on exploring the influence of random injection and discard of edges both demonstrate the robustness of our QGCN model. QGCN remains steady in differently constructed graphs while the baseline model declines dramatically, which indicates that changing the structure of the graph by random injection or discard of edges has almost no influence on the model performance. We ascribe this to the expressive quaternion feature transformation, distinguishing the contribution of different nodes and effectively capturing the graph structural features during message propagation. Thus, it can aggregate more useful information and further lead to better model performance and robustness.

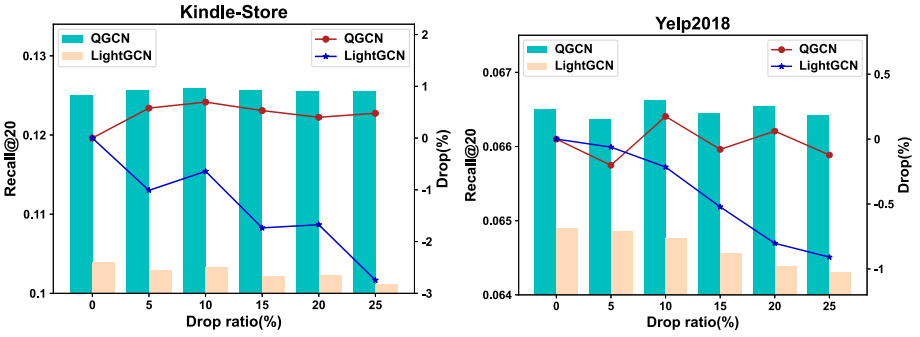

Figure 5 Effect of random edges discard. The bar represents Recall@20, while the line represents the relative performance change compared to the original result

Table 3 Performance of our model and its variants

Dataset	Yelp2018		Amazon-Book		Kindle-Store	
Metric	Recall@20	NDCG@20	Recall@20	NDCG@20	Recall@20	NDCG@20
QGCN-Q	0.0603	0.0491	0.0369	0.0280	0.0939	0.0576
QGCN-W	0.0660	0.0541	0.0485	0.0370	0.1244	**0.0795**
QGCN	**0.0668**	**0.0547**	**0.0489**	**0.0376**	**0.1250**	0.0788

4.5 Ablation study (RQ3)

4.5.1 Influence of components

We perform ablation studies to explore the contribution of different components to the model performance by comparing QGCN with the following two variants:

- **QGCN-Q**: In this variant, we embed all users and items into the real-value space instead of the Quaternion space and maintain the component of feature transformation.
- **QGCN-W**: This variant removes the quaternion transformation matrices during message propagation.

Table 3 shows the results of the two variants of QGCN, and the best results are highlighted in bold. QGCN performs much better than QGCN-Q, which shows the significant influence of modeling in the Quaternion space. And QGCN outperforms QGCN-W in most cases, indicating the effectiveness of quaternion transformation matrices. The comparison between QGCN and its two variants demonstrates that the design of our proposed QGCN model is reasonable and effective.

4.5.2 Influence of Readout Function

Since different pooling methods generate different final user and item embeddings, we conduct experiments and investigate the influence of the readout function applied to our model. Table 4 shows the results under different readout functions, and the best results are highlighted in bold. We can observe that Mean pooling performs relatively better than the other three readout functions, Max, Sum, Concat pooling. We think Mean pooling method could not only maintain the information of nodes but also uniform the user and items representations generated at each layer, leading to more powerful generalization capability.

Table 4 Influence of readout function

Dataset	Yelp2018		Amazon-Book		Kindle-Store	
Metric	Recall@20	NDCG@20	Recall@20	NDCG@20	Recall@20	NDCG@20
Max	0.0501	0.0387	0.0412	0.0312	0.1033	0.0655
Sum	0.0429	0.0541	0.0467	0.0363	0.1206	0.0771
Concat	0.0572	0.0491	0.0475	0.0364	0.1222	0.0776
Mean	**0.0668**	**0.0547**	**0.0489**	**0.0376**	**0.1250**	**0.0788**

4.6 Hyper-parameter study (RQ4)

4.6.1 Effect of dropout rate

Dropout drops the units of the neural networks with a certain probability during the training process, which proves to be an effective way to prevent neural networks from overfitting [10, 27]. Motivated by the previous work applying dropout in graph convolutional network [1] and GCN-based recommendation models [32], we investigate the influence of the dropout rate p ranging from 0.0 to 0.8 on our proposed QGCN model.

Figure 6 displays the experimental results, including Recall@20 and NDCG@20, under different dropout rates on Yelp2018 and Amazon-Book. Specifically, the dropout rate set as 0.1 leads to the best performance. Besides, the performance degrades generally after the peak in that too many neurons lost leads to underfitting and limits the expression of our model. These observations are consistent with the findings of prior effort [32] and demonstrate the effectiveness of proper dropout rate settings in our model.

4.6.2 Effect of regularization

Regularization is an effective strategy to prevent overfitting, so that we tune the coefficient of L_2 normalization λ among $\{1e^{-6}, 1e^{-5}, \ldots, 1e^{-2}\}$ to investigate the influence of the regularization on our proposed model.

Figure 7 shows the performance of our QGCN model under different regularization coefficients λ on Yelp2018 and Amazon-Book. As shown in Figure 7, too small or too large regularization coefficient result in relatively poor performance. Results are relatively steady when the regularization coefficient λ is set between $1e^{-5}$ and $1e^{-4}$, while the performance significantly decrease when λ is set larger than $1e^{-4}$ or smaller than $1e^{-5}$. This indicates that a medium regularization coefficient is more suitable for our model. Specifically, the optimal regularization coefficient for Yelp2018, Amazon-Book, and Kindle-Store is $1e^{-4}$, $1e^{-5}$ and $1e^{-4}$ respectively.

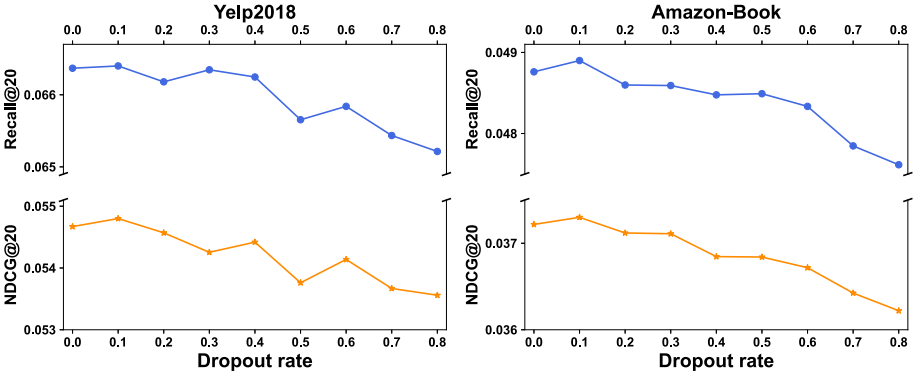

Figure 6 Effect of dropout rate

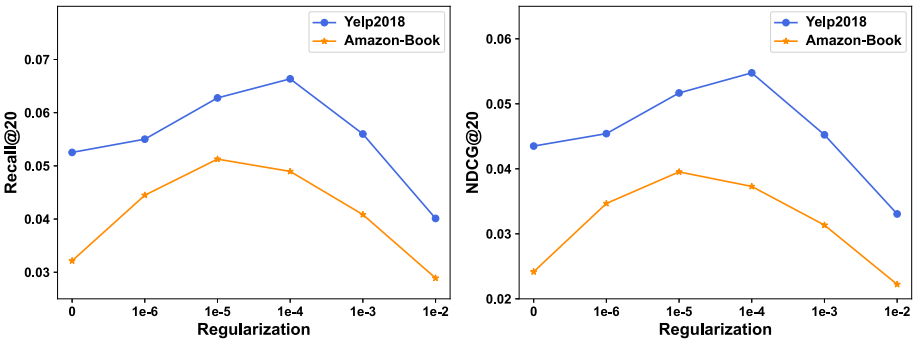

Figure 7 Effect of regularization

5 Related work

5.1 Quaternion-based applications

Quaternion space is a hyper-complex vector space, where each quaternion is a hyper-complex number consisting of one real and three imaginary components. Owing to Hamilton product, which is the multiplication of quaternions, the interactions between real and imaginary components of two quaternions are enhanced, leading to highly expressive computations. In addition, if any slight change happens in the input quaternion, Hamilton product will generate an entirely different output [25] and further influence the final performance. The Quaternion space has been successfully employed in various fields. It has been introduced into various fields for various tasks and achieved remarkable performance improvement than the basic real-valued model. For example, [5] provides the architecture components needed to build deep quaternion networks in the Quaternion space for classification tasks. Zhu et al. [43] re-designs the basic modules like convolution layers and fully-connected layers in the Quaternion space and proposed quaternion-based convolutional neural network (CNN) for image classification and denoising tasks. Parcollet et al. [24] applies the Quaternion space into recurrent neural network (RNN) and long-short term memory neural network (LSTM) for the task of automatic speech recognition. [23] integrates multiple feature views in quaternion-valued CNN to be used for sequence-to-sequence mapping with the CTC model. Parcollet et al. [22] investigates quaternion-based CNN and RNN for speech recognition. Tay et al. [28] proposes quaternion-based attention models and Transformer for NLP tasks. Moreover, there has been some work introducing the Quaternion space into graph representation learning to obtain more expressive graph-level representation [19, 20, 38]. For example, [19] generalizes graph neural networks within the Quaternion space for graph classification, node classification, and text classification. Nguyen et al. [20] Zhang et al. [38] introduce more expressive quaternion representations to model entities and relations for knowledge graph embeddings for knowledge graph completion.

5.2 Collaborative filtering and graph-based recommendation

Collaborative Filtering (CF) based models [8, 14, 34, 36] have shown great performance in learning user and item representations. For example, Matrix Factorization [13] represents users and items with embedding vectors and models the user-item interactions with the inner

product. Neural collaborative filtering [8] utilizes nonlinear neural networks with multiple hidden layers to capture the user-item interactions for better user and item representations. Another research line exploits the user-item interaction graph for recommendation. Prior efforts like ItemRank [7], adopt label propagation on the graph and encourage connected nodes to have similar labels. HOP-Rec [37] firstly performs random walks to augment a user's interactions to exploit the connectivity information.

Recently, Graph Neural Networks(GNN), such as GCN [12] and GAT [29], have shown promising performance in various fields. Moreover, there are approaches that focus on addressing some of the flaws of GNN. API-GNN [40] reduces the disturbance of neighborhood aggregationand. BGN [31] improves the efficiency and scalability of GNN by a binarized graph neural network. Specifically, GCN-based recommendation models have surged to learn better user and item representations in user-item bipartite graphs. For example, GC-MC [1] explores the first-order connectivity between users and items by utilizing only one convolution layer in the user-item bipartite graph. NGCF [32] leverages the message-passing mechanism to obtain high-order connectivity and collaborative signal between users and items. LightGCN [9] simplifies the NGCF [32] model, removing the components of feature transformation and nonlinear activation, improving training efficiency and generation ability. Apart from these GCN-based recommdation models , some methods leverage self-supervised learning for additional information [33, 35], and some research on the heterogeneous graph, opposite of the homogeneous graph [21].

6 Conclusion

In this work, we argued the limitation of the unreasonable operation of removing the feature transformation and modeling users and items in the Euclidean space and performed empirical studies to justify this argument. We moved beyond the Euclidean space, fully utilized the Quaternion space, a hyper-complex space, and proposed a simple yet effective Quaternion-based Graph Convolution Network model formed by a Quaternion Embedding Layer, Quaternion Embedding Propagation Layers, and a Prediction Layer. Extensive experiments on three public benchmark datasets were conducted to evaluate the effectiveness of our proposed model. Results showed that our model outperforms the state-of-the-art methods by a large margin. This indicates that it can better learn user and item representations. Besides, further robustness analysis demonstrated that our QGCN model is more robust to noisy and incomplete graphs and can effectively capture the graph structural features. Moreover, the specific performance comparison showed that our QGCN model gains huge performance improvement on sparse graphs, which is of great significance to the practical applications and real recommendation scenarios.

This work represents an attempt to explore the Quaternion space to model users and items and the effectiveness of quaternion transformation in the Quaternion-based GCN collaborative filtering methods. We believe the insights in this study are enlightening for introducing the Quaternion space into other recommendation scenarios and digging into the nature and effectiveness of quaternion transformation.

Acknowledgements This research was partially supported by the NSFC (61876117, 62176175), the major project of natural science research in the Universities of Jiangsu Province (21KJA520004), the Suzhou Science and Technology Development Program (SYC2022139), the Priority Academic Program Development of Jiangsu Higher Education Institutions, and the Exploratory Self-selected Project of the State Key Laboratory of Software Development Environment.

Author Contributions Yaxing Fang is responsible for the data collection, experiments, and manuscript writing. Pengpeng Zhao, Yanchi Liu, Victor S. Sheng helped review and revise the manuscript a lot. Other authors participated in the review process as well.

Funding This research was partially supported by the NSFC (61876117, 62176175), the major project of natural science research in the Universities of Jiangsu Province (21KJA520004), the Suzhou Science and Technology Development Program (SYC2022139), the Priority Academic Program Development of Jiangsu Higher Education Institutions, and the Exploratory Self-selected Project of the State Key Laboratory of Software Development Environment.

Data Availability The datasets can be accessed through the link corresponding to the footnotes of the datasets. Moreover, we list the pre-processed datasets links:
Yelp2018 and Amazon-Book: https://github.com/gusye1234/LightGCN-PyTorch
Kindle-Store: https://github.com/liufancs/IMP_GCN.

Declarations

Competing interests The authors have no relevant financial or non-financial interests to disclose.

References

1. van den Berg, R., Kipf, T.N., Welling, M. (2017) Graph convolutional matrix completion. CoRR arXiv:1706.02263
2. Chami, I., Ying, Z., Ré, C., et al. (2019) Hyperbolic graph convolutional neural networks. In: NeurIPS, pp. 4869–4880
3. Chen, H., Wang, L., Lin, Y., et al. (2021) Structured graph convolutional networks with stochastic masks for recommender systems. In: SIGIR. ACM, pp. 614–623
4. Dai, H., Li, H., Tian, T., et al. (2018) Adversarial attack on graph structured data. In: ICML, Proceedings of machine learning research, vol. 80. PMLR, pp. 1123–1132
5. Gaudet, C.J., Maida, A.S. (2018) Deep quaternion networks. In: IJCNN. IEEE, pp. 1–8
6. Glorot, X., Bengio, Y. (2010) Understanding the difficulty of training deep feedforward neural networks. In: AISTATS, JMLR proceedings, vol. 9. JMLR.org, pp. 249–256
7. Gori, M., Pucci, A. (2007) Itemrank: A random-walk based scoring algorithm for recommender engines. In: IJCAI, pp. 2766–2771
8. He, X., Liao, L., Zhang, H., et al. (2017) Neural collaborative filtering. CoRR arXiv:1708.05031
9. He, X., Deng, K., Wang, X., et al. (2020) Lightgcn: Simplifying and powering graph convolution network for recommendation. In: SIGIR. ACM, pp. 639–648
10. Hinton, G.E., Srivastava, N., Krizhevsky, A., et al. (2012) Improving neural networks by preventing co-adaptation of feature detectors. CoRR arXiv:1207.0580
11. Kingma, D.P., Ba, J. (2015) Adam: a method for stochastic optimization. In: ICLR (Poster)
12. Kipf, T.N., Welling, M. (2017) Semi-supervised classification with graph convolutional networks. In: ICLR (Poster). OpenReview.net
13. Koren, Y., Bell, R.M., Volinsky, C.: Matrix factorization techniques for recommender systems. Computer **42**(8), 30–37 (2009)
14. Liang, D., Krishnan, R.G., Hoffman, M.D., et al. (2018) Variational autoencoders for collaborative filtering. In: WWW. ACM, pp. 689–698
15. Lin, T., Gao, C., Li, Y. (2019) CROSS: cross-platform recommendation for social e-commerce. In: SIGIR. ACM, pp. 515–524
16. Liu, C., Zhou, C., Wu, J., et al. (2018) Social recommendation with an essential preference space. In: AAAI. AAAI Press, pp. 346–353
17. Liu, F., Cheng, Z., Zhu, L., et al. (2021) Interest-aware message-passing GCN for recommendation. In: WWW. ACM / IW3C2, pp. 1296–1305
18. Liu, Q., Nickel, M., Kiela, D. (2019) Hyperbolic graph neural networks. In: NeurIPS, pp. 8228–8239
19. Nguyen, D.Q., Nguyen, T.D., Phung, D. (2020a) Quaternion graph neural networks. CoRR arXiv:2008.05089
20. Nguyen, D.Q., Vu, T., Nguyen, T.D., et al. (2020b) Quatre: Relation-aware quaternions for knowledge graph embeddings. CoRR arXiv:2009.12517

21. Pang, Y., Wu, L., Shen, Q., et al. (2022) Heterogeneous global graph neural networks for personalized session-based recommendation. In: WSDM. ACM, pp. 775–783
22. Parcollet, T., Ravanelli, M., Morchid, M., et al. (2018a) Speech recognition with quaternion neural networks. CoRR arXiv:1811.09678
23. Parcollet, T., Zhang, Y., Morchid, M., et al. (2018b) Quaternion convolutional neural networks for end-to-end automatic speech recognition. In: INTERSPEECH. ISCA, pp. 22–26
24. Parcollet, T., Ravanelli, M., Morchid, M., et al. (2019) Quaternion recurrent neural networks. In: ICLR (Poster). OpenReview.net
25. Parcollet, T., Morchid, M., Linarès, G.: A survey of quaternion neural networks. Artif. Intell. Rev. **53**(4), 2957–2982 (2020)
26. Rendle, S., Freudenthaler, C., Gantner, Z., et al. (2009) BPR: bayesian personalized ranking from implicit feedback. In: UAI. AUAI Press, pp. 452–461
27. Srivastava, N., Hinton, G.E., Krizhevsky, A., et al.: Dropout: a simple way to prevent neural networks from overfitting. J. Mach. Learn. Res. **15**(1), 1929–1958 (2014)
28. Tay, Y., Zhang, A., Luu, A.T., et al. (2019) Lightweight and efficient neural natural language processing with quaternion networks. In: ACL (1). Association for Computational Linguistics, pp. 1494–1503
29. Velickovic, P., Cucurull, G., Casanova, A., et al. (2018) Graph attention networks. In: ICLR (Poster). OpenReview.net
30. Wang, D., Wang, X., Xiang, Z., et al.: Attentive sequential model based on graph neural network for next poi recommendation. World Wide Web **24**(6), 2161–2184 (2021)
31. Wang, H., Lian, D., Zhang, Y., et al.: Binarized graph neural network. World Wide Web **24**(3), 825–848 (2021)
32. Wang, X., He, X., Wang, M., et al. (2019) Neural graph collaborative filtering. In: SIGIR. ACM, pp. 165–174
33. Wu, J., Wang, X., Feng, F., et al. (2021) Self-supervised graph learning for recommendation. In: SIGIR. ACM, pp. 726–735
34. Wu, Y., DuBois, C., Zheng, A.X., et al. (2016) Collaborative denoising auto-encoders for top-n recommender systems. In: WSDM. ACM, pp. 153–162
35. Xia, L., Huang, C., Xu, Y., et al. (2022) Hypergraph contrastive collaborative filtering. In: SIGIR. ACM, pp. 70–79
36. Xue, H., Dai, X., Zhang, J., et al, (2017) Deep matrix factorization models for recommender systems. In: IJCAI. ijcai.org, pp. 3203–3209
37. Yang, J., Chen, C., Wang, C., et al. (2018) Hop-rec: high-order proximity for implicit recommendation. In: RecSys. ACM, pp. 140–144
38. Zhang, S., Tay, Y., Yao, L., et al. (2019) Quaternion knowledge graph embeddings. In: NeurIPS, pp. 2731–2741
39. Zhou, Y., Wu, J., Chan, T.H., et al.: Interpreting video recommendation mechanisms by mining view count traces. IEEE Trans. Multim. **20**(8), 2153–2165 (2018)
40. Zhou, Y., Shang, Y., Cao, Y., et al.: API-GNN: attribute preserving oriented interactive graph neural network. World Wide Web **25**(1), 239–258 (2022)
41. Zhu, D., Zhang, Z., Cui, P., et al. (2019a) Robust graph convolutional networks against adversarial attacks. In: KDD. ACM, pp. 1399–1407
42. Zhu, Q., Zhou, X., Song, Z., et al. (2019b) DAN: deep attention neural network for news recommendation. In: AAAI. AAAI Press, pp. 5973–5980
43. Zhu, X., Xu, Y., Xu, H., et al. (2018) Quaternion convolutional neural networks. In: ECCV (8), Lecture notes in computer science, vol. 11212. Springer, pp. 645–661

Authors and Affiliations

Yaxing Fang[1] · Pengpeng Zhao[1] · Guanfeng Liu[2] · Yanchi Liu[3] · Victor S. Sheng[4] · Lei Zhao[1] · Xiaofang Zhou[5]

Yaxing Fang
yxfang2020@stu.suda.edu.cn

Guanfeng Liu
guanfeng.liu@mq.edu.au

Yanchi Liu
yanchi.liu@rutgers.edu

Victor S. Sheng
victor.sheng@ttu.edu

Lei Zhao
zhaol@suda.edu.cn

Xiaofang Zhou
zxf@cse.ust.hk

1 School of Computer Science and Technology, Soochow University, Suzhou, China

2 Macquarie University, Sydney, Australia

3 Rutgers University, New Jersey, USA

4 Department of Computer Science, Texas Tech University, Lubbock, USA

5 The Hong Kong University of Science and Technology, Hong Kong SAR, China