Hindawi
Journal of Advanced Transportation
Volume 2022, Article ID 3843021, 12 pages
https://doi.org/10.1155/2022/3843021

Research Article

Research on Recommendation Algorithm of Joint Light Graph Convolution Network and DropEdge

Haicheng Qu, Jiangtao Guo, and Yanji Jiang

College of Software, Liaoning Technical University, Liaoning, Huludao 125105, China

Correspondence should be addressed to Jiangtao Guo; 471921121@stu.lntu.edu.cn

Received 16 October 2021; Accepted 22 December 2021; Published 27 January 2022

Academic Editor: JingXin Dong

Overfitting in a deep neural network leads to low recommendation precision and high loss. To mitigate these issues in a deep neural network-based recommendation algorithm, we propose a recommendation algorithm, LG-DropEdge, joint light graph convolutional network, and the DropEdge. First, to reduce the cost of data storage and calculation, we initialize user and item embedding in the embedding layer of the algorithm. Then, to obtain high-order interaction relationships to optimize the embedding representation, we enrich the embedding by injecting high-order connectivity relationships in the convolutional layer. In the training phase, DropEdge is used to randomly discard connected relationships (interaction edges) to prevent overfitting. Finally, to reasonably aggregate the embedding results learned on all layers, the weighted average is expressed as the final embedding, so that users can make preferences in the item. We conduct experiments on three public datasets, using two performance indicators; namely, recall and NDCG, are used for evaluation. For the Gowalla dataset, compared with the optimal baseline method, recall@20 and ndcg@20 increased by 2.53% and 2.39%, respectively. For the Yelp2018 dataset, recall@20 and ndcg@20 increased by 6.17% and 5.58%, respectively. For the Amazon-book dataset, recall@20 and ndcg@20 increased by 4.82% and 4.67%, respectively. The results show that LG-DropEdge can not only reduce the degree of neural network overfitting but also improve the recommended results' precision.

1. Introduction

Personalized recommendation is a common recommendation method that has been widely used in social media, advertising, e-commerce [1], and other online services. It effectively alleviates the difficulties involved in a user obtaining personalized content due to the explosive growth of information. Its goal is to estimate the likelihood of users to adopt a product based on historical interaction behaviors, such as purchases and clicks. Thus, many studies have focused on recommendation algorithms that are in line with user needs and are practical.

Collaborative filtering (CF) is a method used to build a large-scale recommendation system. Its advantages are strong interpretability and high maturity. The general idea of the algorithm is to predict the items that users may be interested in by analyzing the interaction between users and items. At a high level, the similarity measure between users is based on user rating history, so that ratings from like-minded users can be used to predict the ratings of interested users; it can also be based on the ratings of users who were interested in the past. To realize this idea, a common approach is to reconstruct historical interactions by parameterizing users and items and then predict user preferences based on parameters [2]. Traditional CF recommendation algorithms can be grouped into two categories: neighborhood-based CF methods [3, 4] and matrix factorization (MF) [5–7] methods. MF have been studied more extensively. It is usually assumed that the scoring matrix can be approximated by two lower triangular matrices. On the basis of matrix factorization, probabilistic MF (PMF) optimizes the maximum likelihood by minimizing the mean square error between the observed items and the reconstructed level [5]. Biased MF improves PMF by merging user and item specific deviations and global deviations [6]. In practical applications, the CF method has some problems, such as a

learning process not being displayed, the high-level interaction information between the user and the item not being considered, and the implicit relationship between the user and the item being ignored.

In recent years, the deep-learning algorithm represented by Convolutional Neural Networks (CNNs) has made great progress in many aspects [8, 9], but its design mostly uses regular Euclidean data (which can be expressed in the form of a sequence or a two-dimensional grid), such as image, voice, and natural language. CNNs are suitable for Euclidean data [10] but have limitations in networks with non-Euclidean structures. This is an issue because not everything can be represented as a sequence or two-dimensional grid, such as social networks or chemical molecules. As these data can be regarded as special cases of graph-structured data, researchers naturally think of generalizing CNN to graphs.

2. Related Research

2.1. Graph Convolutional Network. A graph convolutional network (GCN) [11] can overcome the problems of CNNs only being applicable to Euclidean data and can capture the characteristics of a network structure. Development has been rapid and research in this direction is generally divided into two categories: methods based on spectral decomposition and methods based on spatial structure. Spectral decomposition methods mainly deal with the spectral domain of the graph. A spectral network [12] defines the convolution operation in the Fourier domain by calculating the feature decomposition of the Laplacian matrix of the graph, but this convolution operation will cause the convolution kernel not to meet the locality. Henaff et al. proposed introducing a parameter term with smooth coefficients to solve the locality problem of the convolution kernel [13]. Defferrard et al. proposed ChebNet, using K-order convolution to define the graph; the convolution can avoid the calculation of redundant Laplacian matrix eigenvectors [14]. Kipf and Welling proposed reducing the convolution operation to first order [15], which greatly reduces the calculation of graph convolution. As a simplification of spectral decomposition methods, the GCN was formally proposed. To learn the implicit relationship between different nodes, Li et al. proposed a residual Laplacian that an adaptive graph convolution network should be added to the original graph [16]. Methods based on spatial structure mainly deal with graphs of different structures and directly define the convolution operation on the graph. GraphSAGE (Graph Sample and AggreGatE) is a method to generate the embedding vector of the target vertex by learning a function that aggregates the representation of neighbor nodes and calculates the node representation inductively [17]. Unlike earlier methods, the graph attention network (GAT) innovatively uses a self-attention mechanism to provide different weights for the heterogeneity of different nodes [18]. Based on GAT, the heterogeneous graph attention network (HAT) refined two attention mechanisms, namely, node-level attention and semantic-level attention [19].

Due to the powerful expressive ability of graphs, graph-based recommendation algorithms have become one of the most popular research methods. The goal is to reorganize the interactive data into a user-item bipartite graph and use the high-level connectivity between users and items to enrich its representation. PinSage [20] uses local convolution to mark the nodes of the graph-structured data and uses multiple convolution modules to aggregate the local neighborhood features of the nodes to generate node embeddings. Graph Convolutional Matrix Completion (GC-MC) [21] applies a GCN to the user-item graph but only uses the signal of the first-order neighbor. Through random browsing on the graph, Hop-Rec [22] combines matrix decomposition and a graph structure from the neighborhood of each user to obtain high-level information from items. Neural Graph Collaborative Filtering (NGCF) proposed by Wang et al. [23] encodes collaborative signals hidden in user-item interactions and spreads and embeds them in bipartite graphs to achieve high-level neighborhood aggregation. However, its direct inheritance of GCN makes the design quite substantial with high algorithm complexity and difficult algorithm training. Based on the improvement of the traditional GCN, a series of new algorithms have been proposed. Simple Graph Convolution (SGC) [24] eliminates the GCN layer and employs a nonlinear relation between time and a linear algorithm to reduce the complexity of the algorithm.

2.2. LightGCN Algorithm. The previous works that combine GCNs with recommendation mostly inherit GCN and increase its generalization ability. Based on an in-depth analysis of GCN, from the perspective of simplifying GCN design to make it more concise and more suitable for recommendation algorithms, a new algorithm was proposed: light graph convolutional network (LightGCN) [25]. It uses only the GCN as the foundation for neighborhood aggregation. It has been verified via thorough ablation experiments on the special transformation and nonlinear activation of NGCF inherited from the GCN. The study concluded that the two operations inherited from GCN, nonlinear activation and feature transformation, do not make a positive contribution to NGCF. Furthermore, removing them can significantly improve recommendation precision, showing that certain operations in the GCN bring no benefit to the recommendation task and reduce the effectiveness of the algorithm. Therefore, in LightGCN, each user (or item) is first associated with ID embedding, which is then propagated on the user-item interaction graph to enrich its representation. Finally, the embeddings and weights learned in different layers are combined for recommendation prediction. This not only makes the entire algorithm concise in structure, but also gives a great performance improvement compared to other recommendation methods.

2.3. DropEdge. DropEdge [26] involves randomly removing a certain number of edges from the input graph in each training phase to address overfitting and oversmoothing. Overfitting occurs due to the use of a parameter algorithm to fit the distribution of limited training data. The learned

algorithm fits the training data well but is not suitable for the test data. Overfitting weakens the generalization ability of small dataset. Oversmoothing isolates the output representation from the input features as the network depth increases, thus hindering algorithm training. DropEdge can be regarded as a data enhancement technique that increases the randomness and diversity of input data, thereby better preventing overfitting. It can also be regarded as reducing message passing. Losing some edges makes the node connections sparser, which avoids oversmoothing to a certain extent when the GCN is deep.

In addition, DropEdge is different from Dropout [27] and DropNode [28]. Dropout disrupts the feature matrix by randomly setting the feature dimension to zero, but because it does not change the adjacency matrix, it does not have an obvious impact on the overfitting problem. DropNode samples the subgraph and is used for small batch processing. Its principle is that discarding certain nodes can be understood as a special kind of edge discarding. DropNode is for nodes, and edge discarding is achieved indirectly by discarding nodes, whereas DropEdge is edge-oriented and can retain the characteristics of all nodes. That is, with DropEdge, the node only loses the interaction with a certain node, but this does not affect the interaction between the node and other nodes, providing greater flexibility and wider applicability.

Therefore, we propose a joint LightGCN and DropEdge recommendation algorithm named LG-DropEdge. This algorithm is based on the concept of the LightGCN algorithm and integrates the DropEdge which slightly improves the algorithm prediction.

The main contributions of this paper are as follows: (1) we proposed new hybrid recommendation algorithm (2) adding DropEdge to the GCN to enrich input and reduce message passing and (3) changing the final representation of LightGCN from the original average of each layer to a weighted average. Experiments on multiple public datasets verified its advantages and performance.

3. Methods

This section describes the proposed LG-DropEdge. The algorithm can be divided into two parts: a light graph convolution algorithm, which is the basic core part of the algorithm and DropEdge to mitigate overfitting and oversmoothing problems caused by deep networks.

3.1. LG-DropEdge Algorithm. The recommendation algorithm based on the GCN uses the topological structure of the graph to spread and aggregate the information of neighboring nodes and learn the embedding of nodes. The algorithm structure is shown in Figure 1. It is usually divided into three layers: an embedding layer, a convolution layer, and a prediction layer. LightGCN is based on NGCF [23]. Ablation experiments show that the two operations inherited from GCN feature transformation and nonlinear activation do not bring any benefits but negatively impact algorithm training by increasing difficulty. Removing them can significantly improve precision. This reflects that adding useless operations to the target task in GCN does not bring any benefits and reduces effectiveness.

3.1.1. Embedding Layer. In the LG-DropEdge, the main tasks of the embedding layer are to express the entities (users and items) and relationships in the user-item interaction diagram as low-dimensional vectors and to retain all the information of the interaction diagram, which can reduce data storage and calculation costs and filter out some noise data. Following the mainstream recommendation algorithm [23], the IDs of users and items are mapped to vectors using one-hot encoding, denoted by $v_u \in R^h$ and $v_u \in R^h$, where h is the embedding size. Here, $v_u^{(0)}$ and $v_i^{(0)}$ are the initial vectors of user embedding and item embedding, respectively. The number of users u is N and the number of items i is M.

3.1.2. Convolutional Layer. After obtaining the embedding representation of the user (item) node, based on the graph neural network message passing method rule [23], the collaborative signal is obtained on the interactive graph structure and the embeddings of the user and item are optimized. This mainly involves the construction of signals and the aggregation (update) of node embedding. The advantage lies in the embedded representation that can be displayed to associate users and items with high-level collaboration information. This section shows the embedding learning process of the first-order signal and the higher-order signal extended from the first-order signal.

(1) First-Order Signal. The consumer interaction between the user and the item can be used as a characteristic of the user and can be regarded as a collaborative signal of two users.

Structure of the signal: for users and items (u, i) that have an interactive relationship, the resulting signal is defined as

$$S_{u \leftarrow i} = f\left(v_u, v_i, d_{ui}\right), \tag{1}$$

where $S_{u \leftarrow i}$ is the propagated signal, v_u and v_i are embedded inputs, and d_{ui} is the attenuation coefficient of each propagation on the control (u, i). Finally, $f(\cdot)$ is the signal encoding function, expressed as

$$S_{u \leftarrow i} = \frac{1}{\sqrt{\left|N_u\right|\left|N_i\right|}} v_i, \tag{2}$$

where d_{ui} is set to the graph Laplacian norm $1/\sqrt{|N_u||N_i|}$ and N_u and N_i denote the first-order neighbor sets of the user u and the item i, respectively.

Aggregation of node embedding: this is used to enrich the form of u embedding by summarizing the signals of neighbors near the user node u. The aggregate function is defined as

$$v_u^{(1)} = \sum_{i \in N_u} \frac{1}{\sqrt{\left|N_u\right|\left|N_i\right|}} S_{u \leftarrow i}, \tag{3}$$

where $v_u^{(1)}$ represents the embedded representation of the user u obtained after the first embedding and propagation.

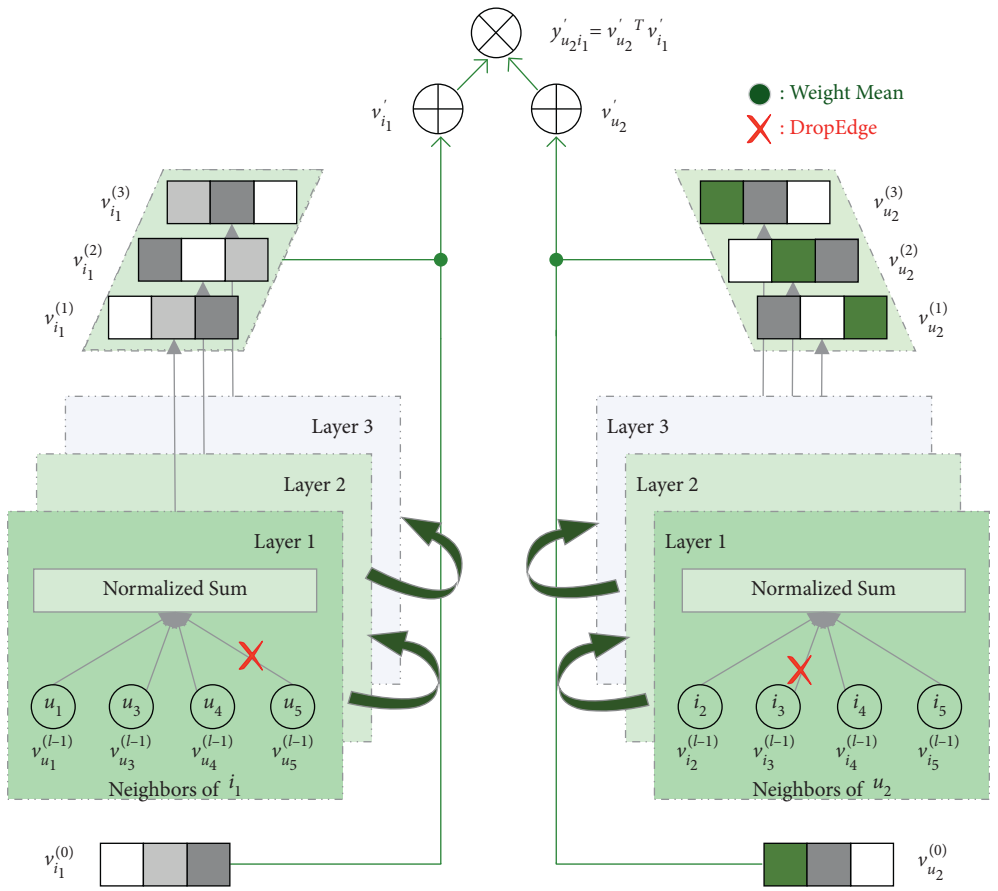

Figure 1: LG-DropEdge algorithm structure.

The self-connection $S_{u\leftarrow u}$ of u is not considered here. Similarly, the representation of the item obtained through embedding and propagation can be obtained.

(2) High-Order Signals. Through the improvement of first-order signals, multiple layers of embedding can be stacked to represent high-order collaborative signals. This high-level signal enables better interpretability in estimating the correlation between users and items.

Signal structure: enrich the signal by stacking multiple layers of embedding, defined as follows:

$$S_{u\leftarrow i}^{(l)} = \frac{1}{\sqrt{|N_u||N_i|}} v_i^{(l-1)}. \tag{4}$$

Node embedding aggregation: aggregate multilayer signals, which can receive signals propagated from layer neighbors, are defined as follows:

$$v_u^{(l)} = \sum_{i\in N_u} \frac{1}{\sqrt{|N_u||N_i|}} S_{u\leftarrow i}^{(l-1)}, \tag{5}$$

where $S_{u\leftarrow i}^{(l-1)}$ is the representation generated from the previous signal transfer step, which is used to store collaborative signals from $(l-1)$ layer neighbours. Similarly, the l layer representation of the item i can be obtained.

3.1.3. Prediction Layer. After spreading the l layer, the l representation of the user u, namely, $\{v_u^{(1)}, v_u^{(2)}, \ldots, v_u^{(l)}\}$, can be obtained. The embeddings obtained at each layer are further combined to form the final representation of the user (item):

$$\begin{aligned} v_u' &= \frac{\sum_{l=0}^{L} \alpha_l v_u^{(l)}}{\sum_{l=0}^{L} \alpha_l} \\ v_i' &= \frac{\sum_{l=0}^{L} \alpha_l v_i^{(l)}}{\sum_{l=0}^{L} \alpha_l}, \end{aligned} \tag{6}$$

where $\alpha_l > 0$ represents the importance of layer embedding in forming the final embedding, which is equivalent to an attention mechanism. This is because, when the number of layers increases, the weight is on a downward trend. The final representation is changed from the original average of each layer to a weighted average. The deeper the layer, the greater the weight, which emphasizes the importance of the deep signal. To prevent the algorithm becoming too complex, α_l is set to $(1/k+1)$.

Finally, the inner product is used to estimate the user's preference for the item:

$$y_{ui}' = v_u'^T v_i'. \tag{7}$$

It can be used as the ranking score generated by the recommendation.

3.2. DropEdge Module. The main task of DropEdge [26] is to randomly lose a certain number of edges from the input graph at each training phase, which can be understood as data enrichment. This can increase the randomness and diversity of the input data, which can mitigate overfitting. DropEdge's approach can also be understood as the simplification of data transmission. Losing some edges makes node connections sparser, which can effectively avoid oversmoothing.

To apply the DropEdge, (5) is transformed into a matrix form, as follows:

$$V^{(l)} = \mathrm{L}V^{(l-1)}, \tag{8}$$

where $V^{(l)} \in R^{(N+M)\times d}$ is the representation of users and items obtained after embedding in the propagation layer and d is the embedding size. The initial signal $V^{(0)}$ is set to V; that is, $v_u^{(0)} = v_u$ and $v_i^{(0)} = v_i$. Finally, L is the Laplacian matrix (symmetrical normalization) of the user-item interaction graph, which is defined as

$$\begin{aligned} L &= D^{-(1/2)}\mathrm{A}\mathrm{D}^{-(1/2)} \\ A &= \begin{bmatrix} 0 & R \\ R^T & 0 \end{bmatrix}, \end{aligned} \tag{9}$$

where A is the adjacency matrix, $D \in R^{(M+N)\times(M+N)}$ is the pair angle matrix, $D_{ii} = |N_i|$ is the number of nonzero entries in the l-th row vector of the adjacency matrix A, and $R \in R^{M\times N}$ is the user-item interaction matrix. Finally, the final embedding matrix used for algorithm prediction is

$$\begin{aligned} V &= \alpha_0 V^{(0)} + \alpha_1 V^{(1)} + \cdots + \alpha_L V^{(L)} \\ &= \alpha_0 V^{(0)} + \alpha_1 \mathrm{L}V^{(0)} + \cdots + \alpha_L L^{(L)} V^{(0)}. \end{aligned} \tag{10}$$

In each training phase, DropEdge randomly selects a certain percentage of edges of the input graph. In other words, it randomly sets Vp nonzero elements in the adjacency matrix A to zero, where V is the total number of edges, and p is the loss rate. The obtained adjacency matrix A_{drop} is expressed as

$$A_{\mathrm{drop}} = A - A', \tag{11}$$

where A' is expanded from a random subset of the original edge size of Vp. Then, the matrix is normalized and expressed as $\widehat{A}_{\mathrm{drop}}$, which is used to replace A in the convolution operation in (9).

3.3. Algorithm Training. Reference algorithm [2], using paired Bayesian Personalized Ranking (BPR) loss [29], considers the relative order between observed and unobserved user-item interactions and encourages the prediction of observed items to be higher than that of unobserved items; the interactions with corresponding observed items are more reflective of user preferences.

The loss function of the algorithm is

$$\mathrm{Loss} = -\sum_{(u,i,j)\in O} \ln \sigma\left(y'_{ui} - y'_{uj}\right) + \lambda \left\| V^{(0)} \right\|_2^2, \tag{12}$$

where $O = \{(u,i,j) | (u,i) \in R^+, (u,j) \in R^-\}$ is the paired training data, R^+ is the observed interaction, R^- is the unobserved interaction, $\sigma(\cdot)$ is a sigmoid function, λ controls the intensity of L_2 regularization, and the training parameter is only the embedding of the 0th layer; that is, $\Theta = \{V^{(0)}\}$. A small batch of the Adam [30] optimizer is used to predict the algorithm and update the algorithm parameters.

4. Experiments

To verify the recommended performance of the proposed LG-DropEdge, this study used the PyTorch deep-learning framework. The operating system used in the experiment was Windows 10, the graphics card was a Nvidia Titan V, and the CPU was an i7-8700K. The Python version was 3.6. Experiments and analysis were conducted using the Pycharm2020 development tool and PyTorch deep-learning framework.

To compare the performance of the proposed algorithm with other algorithms and verify the effectiveness of the algorithm's own modules, we designed three sets of experiments:

(Q1) Compared with the most advanced LightGCN, how is the performance of LG-DropEdge compared?

(Q2) How do the settings of different improved modules affect the performance of LG-DropEdge?

(Q3) How do the hyperparameter settings (such as edge loss rate) impact the effectiveness of LG-DropEdge?

4.1. Experimental Dataset. To evaluate the effectiveness of LG-DropEdge, we conducted experiments on three public datasets: Gowalla, Yelp2018, and Amazon-book. The Gowalla dataset uses the login dataset from Gowalla [31], where users share their location by logging in. Yelp2018 comes from the 2018 version of the Yelp Challenge, in which companies such as hotels and coffee shops were used as items. Amazon-book [32] comes from book data in Amazon reviews. There are differences in data entities, numbers of interactions, and sparsity, which can meet the needs of different data characteristics as required by the algorithm. Table 1 summarizes the statistics of the three datasets.

Inspired by mainstream recommendation algorithms [23, 25], for each dataset, 80% of each user's historical interactions were randomly selected to form the training set, and the rest were used as the test set. From the training set, 10% of the interactions were randomly selected as the validation set to adjust the hyperparameters. Each observed user-item interaction was treated as a positive instance. A negative sampling strategy was then used to pair it with a negative item that the user had not previously consumed.

Table 1: Statistics of the tested datasets.

Dataset	Gowalla	Yelp2018	Amazon-book
User	29,858	31,668	52,643
Item	40,981	38,048	91,599
Interaction	1,027,370	1,561,406	2,984,108
Density	0.000,84	0.001,30	0.000,62

4.2. Experimental Setup

4.2.1. Evaluation Index. The performance assessment of the recommendation algorithm has many aspects, divided into a quantitative calculation and qualitative description. This section describes the performance of the algorithm from the perspective of prediction precision. When calculating this indicator, an offline dataset is required that contains historical user behavior data. The dataset is then divided into a training set and a test set, and finally the user's behavior on the test set is predicted by establishing a user's behavior and interest algorithm on the training set. The coincidence degree of the predicted behavior and the actual behavior on the test set are calculated as the prediction precision. For users' Top-N recommendations, algorithm [14, 29] used evaluation indicators, such as precision, recall, and Normalized Discounted Cumulative Gain (NDCG) for $N = \{20, 100\}$.

Let $R(u)$ denote the recommendation list given to the user through training and let $T(u)$ denote the behavior list on the test set.

The precision represents the proportion of the number of samples that are correctly predicted to the total number of samples and is defined as

$$\text{precision@}N = \frac{\sum_{u \in N} |R(u) \cap T(u)|}{\sum_{u \in N} |T(u)|}. \tag{13}$$

The recall of the recommended result represents the probability that the sample is correctly predicted to occupy the actual sample, which is defined as

$$\text{recall@}N = \frac{\sum_{u \in N} |R(u) \cap T(u)|}{\sum_{u \in N} |R(u)|}. \tag{14}$$

The NDCG of the recommended result is defined as

$$\begin{aligned} \text{ndcg@}N &= \frac{\text{dcg}}{\text{idcg}}, \\ \text{dcg} &= \sum_{i=1}^{N} \frac{2^{\text{rel}_i} - 1}{\log_2(i+1)}, \\ \text{idcg} &= \sum_{i=1}^{|\text{REL}_N|} \frac{2^{\text{rel}_i} - 1}{\log_2(i+1)}, \end{aligned} \tag{15}$$

where dcg is the cumulative gain of loss, idcg is the maximum value of dcg under ideal conditions, and $\text{rel}_i \in \{0, 1\}$ indicates the user's rating for the i-th item. Finally, $|\text{REL}_N|$ indicates that the results are sorted in descending order of relevance, where the set consisting of the previous N results is adopted; that is, according to the most, sort the results in an optimal way.

4.2.2. Baseline. To demonstrate the effectiveness of the proposed algorithm, we compared it with the following methods:

NeuMF [2]: this method is an advanced neural collaborative filtering algorithm that uses multiple hidden layers above elements and the concatenation of user and item embeddings to capture their nonlinear characteristic interactions.

Hop-Rec [22]: this method combines matrix decomposition and graphs through random walks on the graph, combined with the degree of the vertices, sampling different positive samples with a certain probability, and assigning attenuation coefficients to the ranking pairs obtained in different orders.

NGCF [23]: this method explicitly introduces the collaborative signal into the collaborative filtering algorithm and achieves this by using the high-level connectivity in the user-item interaction graph.

LightGCN [25]: based on NGCF, this method removes feature changes and nonlinear activation through ablation experiments and adds a weight factor to the final aggregation, which is greatly improved. It is the latest direction of graph convolution.

4.2.3. Parameter Settings. With reference to NGCF [23] and LightGCN [25], the LG-DropEdge algorithm is implemented in PyTorch. Considering the settings of the comparison experiment, choose the same hyperparametric settings as NGCF and LightGCN, etc. To ensure the accuracy and fairness of the comparison results, the embedding size of all algorithms is set to 64; the learning rate is set to $1e^{-3}$; the number of layers is set to 3; the batch size is set to 2048; the regularization coefficient is set to $1e^{-3}$; considering the convergence of loss, the number of training is set to 1500. Grid search is performed on the hyperparameters and the edge loss rate is determined in $\{0.8, 0.6, 0.5, 0.4, 0.2\}$.

4.3. Performance Comparison (Q1)

4.3.1. Performance Comparison with LightGCN. Compared with the LightGCN algorithm, Table 2 records the algorithm performance for different datasets and different indicators and shows the percentage improvement of each indicator. It reveals the clear improvement made by the proposed LG-DropEdge.

In the three sets of comparative experiments, LG-DropEdge performs better than LightGCN. In the three datasets, the algorithm performance on the Yelp2018 dataset has been greatly improved; the precision index has increased by 6.04%

Table 2: Performance comparison of LG-DropEdge and LightGCN.

Dataset	Method	Precision@20	Recall@20	ndcg@20
Gowalla	LightGCN	0.0558	0.1821	0.1545
	LG-DropEdge	**0.0575 (+3.04%)**	**0.1867 (+2.53%)**	**0.1582 (+2.39%)**
Yelp2018	LightGCN	0.0283	0.0632	0.0520
	LG-DropEdge	**0.0301 (+6.36%)**	**0.0671 (+6.17%)**	**0.0549 (+5.58%)**
Amazon-book	LightGCN	0.0171	0.0415	0.0321
	LG-DropEdge	**0.0180 (+5.26%)**	**0.0435 (+4.82%)**	**0.0336 (+4.67%)**

The bold values represent the experimental results of the algorithm proposed in this paper and the improvement effect on the classical experiment.

on average. The main reason for this is that the Yelp2018 datasets are sparser than the other two datasets. A dataset with high sparseness has a large number of user interaction items, and the data characteristics can be better retained after multiple algorithm training phases. In addition, the average precision of the three datasets increased by 4.89%, recall increased by 4.51%, ndcg increased by 4.21%, and overall performance improved.

4.3.2. Overall Comparison. To verify the advantages of the LG-DropEdge recommendation algorithm in terms of precision, we implemented performance comparisons with other classic algorithms (NeuMF [2], Hop-Rec [22], NGCF [23], and LightGCN [25]). The experimental results are shown in Table 3.

The following can be seen from Table 3:

(1) NeuMF performs better than Hop-Rec on the Amazon-book dataset. Since Hop-Rec is implemented by combining MF and graphs, its performance largely depends on the random walk algorithm, and the effect is not very obvious because it does not make full use of hight-order connectivity of graphs.

(2) Since Hop-Rec is implemented by combining MF and graphs, it does not make full use of high-order connectivity. Its performance largely depends on the random walk, and the effect is not very obvious.

(3) NGCF yields a significantly better performance than NeuMF and Hop-Rec because it explicitly introduces the collaborative signal into the system filtering algorithm and spreads it on the interactive graph. However, its algorithm directly inherits GCN, which leads to increased algorithm complexity and training time.

(4) LightGCN, as a simplification of NGCF, yields a powerful performance, but its algorithm does not solve the problem of deep-network overfitting. Further, it uses relatively simple aggregation functions, which limit the effect of improvement.

(5) LG-DropEdge yields the best performance on all datasets, particularly Yelp2018, which showed an increase of more than 5%. Thus, the LG-DropEdge algorithm can be used to improve the precision of a recommendation system.

4.4. Ablation Experiment (Q2). To demonstrate the feasibility of LG-DropEdge, the algorithm is subjected to ablation experiments. To verify the effectiveness of the combined modules, we performed several sets of experiments on the three datasets: removing the improved aggregation function and keeping DropEdge, named LightGCN + DropEdge; removing DropEdge and keeping the improved aggregation function, named LightGCN + f; removing both DropEdge and the improved aggregation function, which is the original LightGCN; and the proposed method, LG-DropEdge.

The results are as shown in Table 4.

The following findings can be made from Table 4:

(1) Compared with LightGCN, precision, recall, and ndcg increased by 2.33%, 1.48%, and 1.23%, respectively, in the LightGCN + f algorithm

(2) Compared with LightGCN, precision, recall, and ndcg decreased by 0.72%, 0.49%, and 0.19% respectively, in the LightGCN + DropEdge algorithm

(3) Compared with LightGCN, precision, recall, and ndcg increased by 3.04%, 2.53%, and 2.39%, respectively, in the LG-DropEdge algorithm

The results are as shown in Table 5.

The following findings can be made from Table 5:

(1) Compared with LightGCN, precision, recall, and ndcg increased by 5.65%, 5.38%, and 5.00%, respectively, in the LightGCN + f algorithm

(2) Compared with LightGCN, precision, recall, and ndcg decreased by 2.12%, 2.58%, and 2.69%, respectively, in the LightGCN + DropEdge algorithm

(3) Compared with LightGCN, precision, recall, and ndcg increased by 6.36%, 6.17%, and 5.58%, respectively, in the LG-DropEdge algorithm

The results are as shown in Table 6.

The following Findings can be made from the Table 6:

(1) Compared with LightGCN, precision, recall, and ndcg increased by 2.92%, 1.69%, and 1.56%, respectively, in the LightGCN + f algorithm

(2) Compared with LightGCN, precision, recall, and ndcg increased by 2.34%, 2.65%, and 2.49%, respectively, in the LightGCN + DropEdge algorithm

(3) Compared with LightGCN, precision, recall, and ndcg increased by 5.26%, 4.82%, and 4.67%, respectively, in the LG-DropEdge algorithm

Table 3: Overall performance comparison.

Dataset method	Gowalla		Yelp2018		Amazon-book	
	Recall@20	ndcg@20	Recall@20	ndcg@20	Recall@20	ndcg@20
NeuMF	0.1339	0.1050	0.0445	0.0359	0.0327	0.0248
Hop-Rec	0.1399	0.1201	0.0517	0.0428	0.0296	0.0211
NGCF	0.1547	0.1313	0.0562	0.0459	0.0324	0.0250
LightGCN	0.1821	0.1545	0.0632	0.0520	0.0415	0.0321
LG-DropEdge	**0.1867**	**0.1582**	**0.0671**	**0.0549**	**0.0435**	**0.0336**

The bold values represent the experimental results of the algorithm proposed in this paper.

Table 4: Performance of the Gowalla dataset under the four algorithms.

Algorithm	Precision	Recall	ndcg
LightGCN	0.0558	0.1821	0.1545
LightGCN + f	0.0571	0.1848	0.1564
LightGCN + DropEdge	0.0554	0.1812	0.1542
LG-DropEdge	0.0575	0.1867	0.1582

Table 5: Performance of the Yelp2018 dataset under the four algorithms.

Algorithm	Precision	Recall	ndcg
LightGCN	0.0283	0.0632	0.0520
LightGCN + f	0.0299	0.0666	0.0546
LightGCN + DropEdge	0.0277	0.0614	0.0506
LG-DropEdge	0.0301	0.0671	0.0549

Table 6: Performance of the Amazon-book dataset under the four algorithms.

Algorithm	Precision	Recall	ndcg
LightGCN	0.0171	0.0415	0.0321
LightGCN + f	0.0176	0.0422	0.0326
LightGCN + DropEdge	0.0175	0.0426	0.0329
LG-DropEdge	0.0180	0.0435	0.0336

Combining Tables 4–6, the following can be obtained:

(1) Compared with LightGCN, LightGCN + f algorithm has different degrees of improvement in the three datasets of precision, recall, and ndcg

(2) Compared with LightGCN, the LightGCN + DropEdge algorithm has decreased in the precision, recall, and ndcg indicators of the Gowalla and Yelp2018 datasets, but there has been a small increase in the Amazon-book dataset. It can be seen that adding DropEdge has less impact on the performance of less sparse datasets

(3) Compared with LightGCN, LG-DropEdge has a greater improvement over LightGCN-f on the three indicators of precision, recall, and ndcg in three datasets, which fully demonstrates that modifying the aggregation function and adding DropEdge make the algorithm more accurate

To explain the effectiveness of the DropEdge, we compared the loss tested under the three datasets at the same ndcg level. The performance on the test set measures the true performance of the algorithm. Under the same test set index (ndcg@20), the training performance was the same, and the train_loss produced by adding DropEdge with the training set was higher, indicating that it is at the same level and there is no mitigation of overfitting. Figure 2 compares the train_loss for 23 sets of data in Gowalla dataset, Figure 3 compares the train_loss for 55 sets of data in Yelp2018 dataset, and Figure 4 compares the train_loss for 41 sets of data in Amazon-book dataset (ndcg is arranged in increasing order), showing that it has increased to different degrees. It can be seen that adding DropEdge is effectively mitigating overfitting. Furthermore, as ndcg increases, LG-DropEdge performs more smoothly in terms of loss, in contrast to the irregular fluctuations of LightGCN. Therefore, the algorithm proposed in this paper has many advantages in mitigating overfitting and oversmoothing problems.

4.5. Hyperparameter Experiment (Q3). To verify the degree of influence of the edge loss rate on the algorithm, the edge loss rate p is determined in {0.8, 0.6, 0.5, 0.4, 0.2} (using three datasets), as in [26].

Figure 5 shows the performance for precision, recall, train_time, and ndcg under different edge loss rates. As seen in Figure 5(a), starting from 0.0, precision and recall maintain the same upward trend as the edge loss rate increases. The first peak is reached at 0.4, after which the two indicators decrease steadily and slightly between 0.4 and 0.5, increase significantly between 0.5 and 0.6, and achieve their best performance at 0.6. After 0.6,

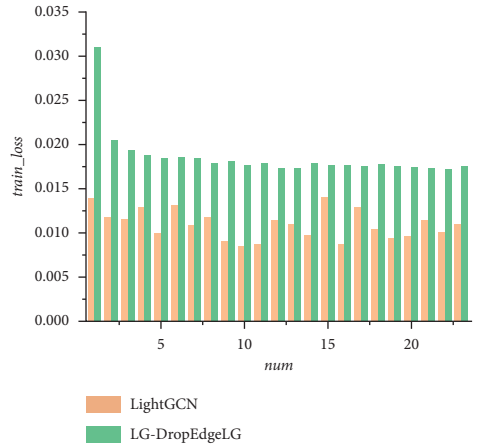

Figure 2: Train_loss comparison at the same ndcg level (Gowalla).

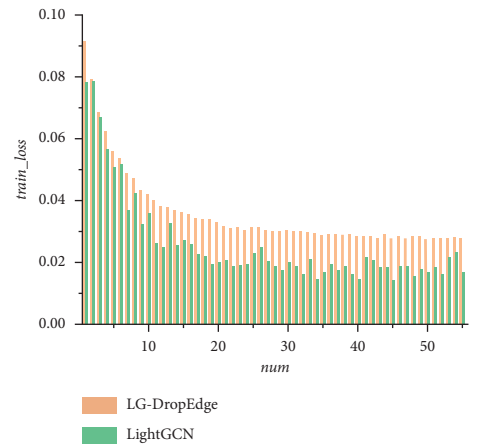

Figure 3: Train_loss comparison at the same ndcg level (Yelp2018).

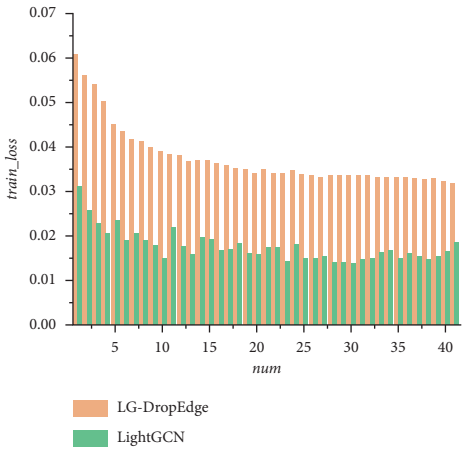

Figure 4: Train_loss comparison at the same ndcg level (Amazon-book).

they begin to decline sharply. The precision index is at its worst value at 0.8. Meanwhile, as seen in Figure 5(b), the ndcg index maintains a steady increase from 0.0, achieves maximum value at 0.6, after which it drops sharply, and reaches its worst value at 0.8. The train_time index starts at the lowest value at 0.0 and then increases rapidly to 0.2, before declining slowly. Therefore, the overall trends for precision, recall, and ndcg are increasing from the start 0.0, achieving a maximum at 0.6, and thereafter declining rapidly. Time is expressed as the average train_time for an epoch. It is obvious that when DropEdge is increased, the train_time increases significantly. Undoubtedly, it increases the difficulty of training. From Figure 5, when the train_time is relatively short, the precision index performs the best and the edge loss rate p of DropEdge is finally determined to be 0.6.

Figure 6 shows the performance for precision, recall, train_time, and ndcg under different edge loss rates. As can be seen from the figure, in Figure 6(a), the precision and recall both increase between 0.0 and 0.2, and they begin to decline between 0.2 and 0.4, increase between 0.5 and 0.6, and then continue to decline. The only trend difference is between 0.4 and 0.5, precision increases, and recall decreases, but the overall trend is the same, first increasing to the highest point ($p = 0.2$) and then slightly lowering, then slightly increasing, and finally falling to the lowest. In Figure 6(b), the ndcg indicator increases between 0.0 and 0.2, reaches the highest point at that time, drops to the lowest point between 0.2 and 0.4, and then increases sharply between 0.4 and 0.5; it

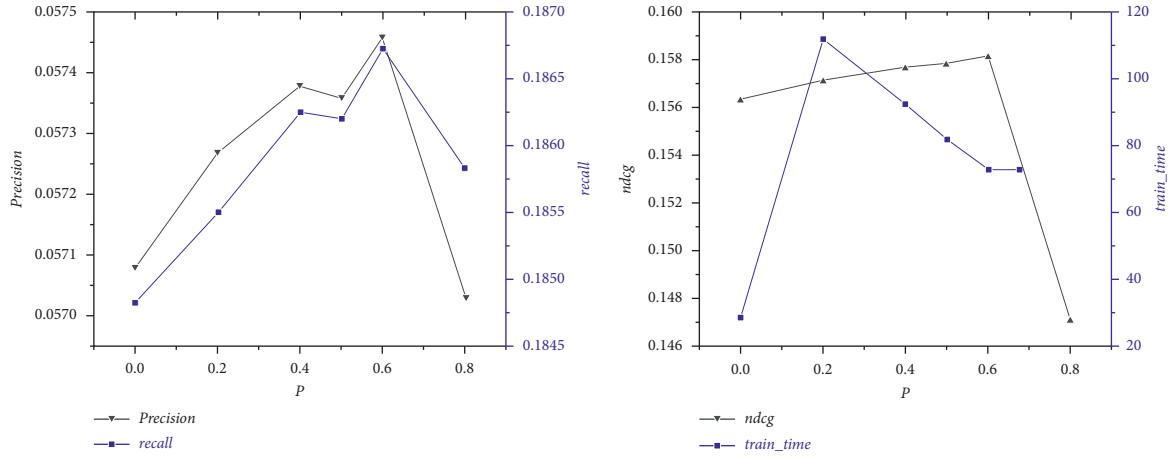

Figure 5: Algorithm performance under different edge drop rates (Gowalla).

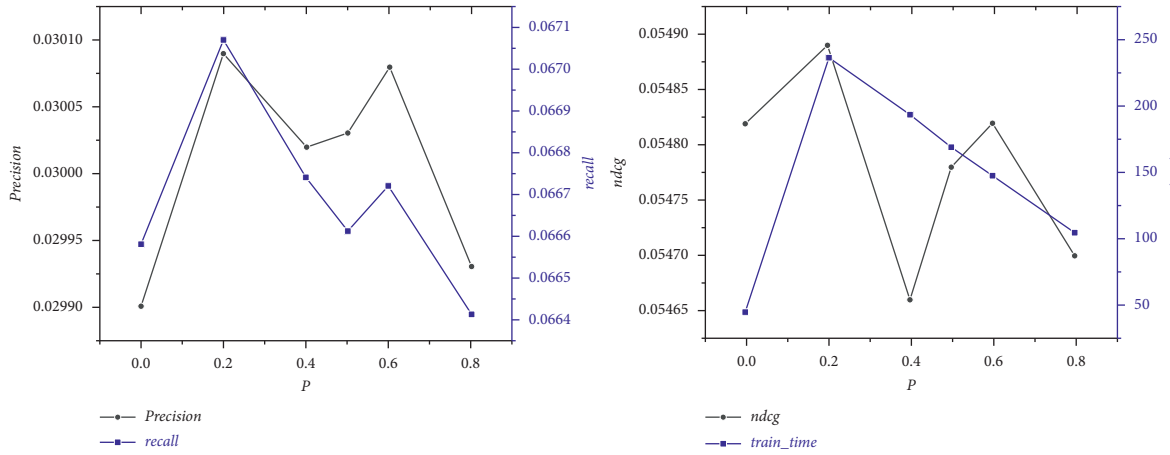

Figure 6: Algorithm performance under different edge drop rates (Yelp2018).

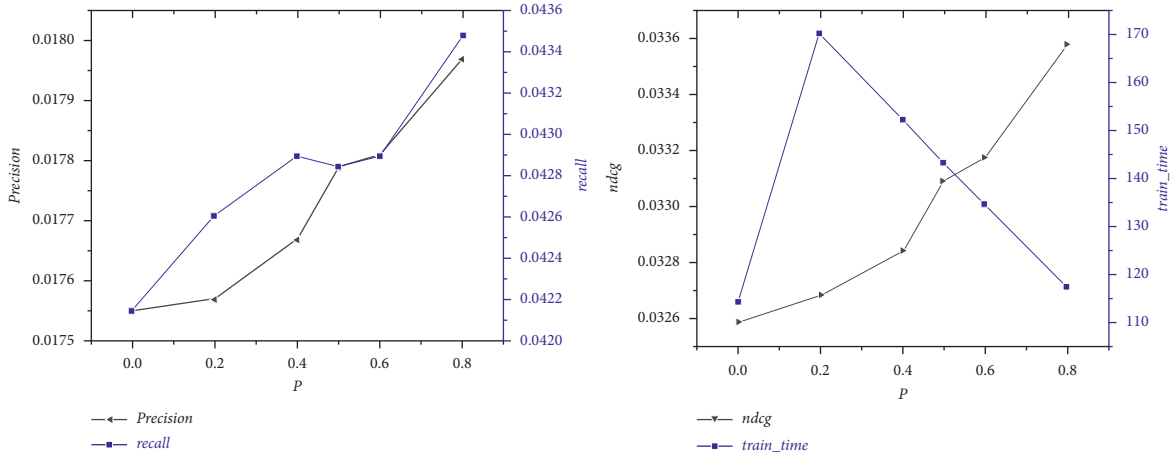

Figure 7: Algorithm performance under different edge drop rates (Amazon-book).

increases slowly between 0.5 and 0.6 and begins to decline after 0.6; while train_time indicator starts to decline after rising from 0.0 to 0.2 to the highest point. Combined with the two subgraphs, all three accuracy indicators reach the best state at that time ($p = 0.2$), then begin to decline, then rise and reach the second peak at $p = 0.6$, and then begin to decline.

According to the rule of relatively little train_time and priority in accuracy index performance, the edge loss rate p of DropEdge is finally determined to be 0.2.

Figure 7 shows the performance for precision, recall, train_time, and ndcg under different edge loss rates. It can be seen from the figure that, in Figure 7(a), the two indicators generally show a continuous increase trend. Only precision has a small decline between 0.4 and 0.5, both of which reach the maximum value at $p = 0.8$; in Figure 7(b), ndcg also shows an increase trend, reaching the maximum value at $p = 0.8$, while train_time at $p = 0.2$ reaches the maximum value and then continues to decline. The two indicators have an intersection between 0.5 and 0.6. After the intersection, they both develop in a positive direction and reach the optimal state of the algorithm at $p = 0.8$, especially after $p = 0.6$ showing a rapid positive growth, so the edge loss rate p of DropEdge is finally determined to be 0.8.

The purpose of this section is to fully verify the improvement of the performance of the algorithm after adding the DropEdge module. It can be seen that the final boundary edge loss rate for different datasets is different. It can be seen that, in machine learning, the algorithm's adaptability to different datasets is different. Based on the results of the above three datasets, there is reason to believe that the performance of arithmetic has been greatly improved by adding the DropEdge module.

5. Conclusion

This paper proposed a joint light graph convolutional network and DropEdge recommendation algorithm (LG-DropEdge). The DropEdge was developed based on the LightGCN framework, which improves the final multilayer fusion aggregation function, mitigates the overfitting problem caused by the deep network algorithm, and improves the precision, recall, and ndcg, and enhances the interpretability of the recommendation algorithm. The proposed algorithm has improved the recommendation accuracy in the offline experiments of the three datasets, but it has not been verified in practical application. In the later research, we will continue to apply other public datasets to our algorithm and gradually apply them to the actual recommendation system to improve the generalization of the algorithm. The existing recommendation algorithm based on graph convolution network generally controls the number of convolution layers at 3 layers considering the attenuation degree of propagation factor, while DropEdge fully demonstrates that its performance is generally in deeper convolutional networks. This paper sets the number of convolution layers to be 3 layers considering the sparseness of datasets and the needs of contrast algorithms. Therefore, future research is to select more sparse datasets for comparative experiments and further analyse the relationship between the number of convolutional layers and the recommendation accuracy.

Data Availability

The data used to support the findings of the study are included in the supplementary files.

Conflicts of Interest

The authors declare that there are no conflicts of interest.

Acknowledgments

This paper was supported by Young Scientists' Fund of National Natural Science Foundation of China and Natural Science Foundation of Liaoning Province.

Supplementary Materials

gowalla-test/train.txt;yelp-test/train.txt;amazon-test/train.txt includes data used to support the findings of this study. run-gowalla/yelp2018/amazon.txt includes data values given in Tables 2–6 and figures, NGCF and LightGCN results, the datasets needed for the experiment, and part of the log files of the experiment. The results of NeuMF and Hop-Rec in Table 3 are quoted from Xiang Wang, Xiangnan he, Meng Wang, Fuli Feng, and Tat Seng Chua, "neural graph collaborative filtering," in SIGIR, pp. 165-174. The code data used to support the findings of this study have been deposited in the [Baidu Netdisk] repository (link: https://pan.baidu.com/s/1JUFyBpoc3RPoKkPsQXvXkQ; extraction code: gaz1). (*Supplementary Materials*)

References

[1] Y. J. Zhang, Z. Dong, and X. W. Meng, "Research on personalized advertising recommendation systems and their applications," *Chinese Journal of Computers*, vol. 44, no. 3, pp. 531–561, 2021.

[2] X. He, L. Liao, H. Zhang, L. Nie, and X. S. Chua, "Neural collaborative filtering," in *Proceedings of the 26th International Conference on World Wide Web*, pp. 173–182, Perth, Australia, April 2017.

[3] D. Valcarce, A. Landin, J. Parapar, and Á. Barreiro, "Collaborative filtering embeddings for memory-based recommender systems," *Engineering Applications of Artificial Intelligence*, vol. 85, pp. 347–356, 2019.

[4] M. Gao, B. Ling, L. Yang, J. Wen, Q. Xiong, and S. Li, "From similarity perspective: a robust collaborative filtering approach for service recommendations," *Frontiers of Computer Science*, vol. 13, no. 2, pp. 231–246, 2019.

[5] A. Mnih and R. R. Salakhutdinov, "Probabilistic matrix factorization," *Advances in Neural Information Processing Systems*, pp. 1257–1264, 2008.

[6] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *Computer*, vol. 42, no. 8, pp. 30–37, 2009.

[7] Y. He, C. Wang, and C. Jiang, "Correlated matrix factorization for recommendation with implicit feedback," *IEEE Transactions on Knowledge and Data Engineering*, vol. 31, no. 3, pp. 451–464, 2019.

[8] Y. Cai, T. Luan, H. Gao et al., "YOLOv4-5D: an effective and Efficient Object Detector for Autonomous Driving," *IEEE Transactions on Instrumentation and Measurement*, vol. 70, pp. 1–13, Article ID 4503613, 2021.

[9] D. Ma, X. Song, and P. Li, "Daily Traffic Flow Forecasting through a Contextual convolutional Recurrent neural network Modeling inter- and Intra-Day Traffic Patterns," *IEEE*

Transactions on Intelligent Transportation Systems, vol. 22, no. 5, pp. 2627–2636, 2021.

[10] F. Monti, D. Boscaini, J. Masci, E. Rodolà, J. Svoboda, and M. Bronstein, "Geometric Deep Learning on Graphs and Manifolds Using Mixture Model CNNs," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5425–5434, Honolulu, HI, USA, July 2017.

[11] F. Scarselli, M. Gori, G. Monfardini, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network Model," *IEEE Transactions on Neural Networks*, vol. 20, no. 1, pp. 61–80, 2009.

[12] J. Bruna, W. Zaremba, A. D. Szlam, and Y. LeCun, "Spectral networks and locally connected networks on graphs," 2014, https://arxiv.org/abs/1312.6203.

[13] M. Henaff, J. Bruna, and Y. LeCun, "Deep Convolutional Networks on Graph-Structured Data," 2015, https://arxiv.org/abs/1506.05163.

[14] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with Fast Localized spectral filtering," *Advances in Neural Information Processing Systems*, vol. 29, pp. 3844–3852, 2016.

[15] T. Kipf and M. Welling, "Semi-Supervised Classification with Graph Convolutional Networks," 2017, https://arxiv.org/abs/1609.02907.

[16] R. Li, S. Wang, F. Zhu, and J. Huang, "Adaptive graph convolutional neural networks," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, LA, USA, February 2018.

[17] W. L. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pp. 1025–1035, Red Hook, NY, United States, December 2017.

[18] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph Attention Networks," 2018, https://arxiv.org/abs/1710.10903.

[19] X. Wang, H. Ji, C. Shi et al., "Heterogeneous Graph Attention Network," in *Proceedings of the World Wide Web Conference*, pp. 2022–2032, CA, San Francisco, USA, May 2019.

[20] R. Ying, R. He, K. Chen, P. Eksombatchai, L. H. William, and J. Leskovec, "graph convolutional neural networks for Web-scale recommender systems," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 974–983, London, United Kingdom, August 2018.

[21] R. V. D. Berg, T. Kipf, and M. Welling, "Graph Convolutional Matrix Completion," 2017, https://arxiv.org/abs/1706.02263.

[22] J. H. Yang, C. M. Chen, C. J. Wang, and M. F. Tsai, "HOP-rec: high-order proximity for implicit recommendation," in *Proceedings of the 12th ACM Conference on Recommender Systems*, pp. 140–144, British Columbia, Vancouver, Canada, October 2018.

[23] X. Wang, X. He, M. Wang, F. Feng, and T. S. Chua, "Neural graph collaborative filtering," in *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval SIGIR'19*, pp. 165–174, Paris, France, July 2019.

[24] F. Wu, T. Zhang, A. Souza, C. Fifty, Y. Tao, and K. Q. Weinberger, "Simplifying Graph Convolutional Networks," pp. 6861–6871, 2019, https://arxiv.org/abs/1902.07153.

[25] X. He, K. Deng, X. Wang, Y. Li, Y. D. Zhang, and M. Wang, "LightGCN: simplifying and powering graph convolution network for recommendation," in *Proceedings of the 43rdInternational ACM SIGIR Conference on Research and Development in Information Retrieval, Virtual Event*, pp. 639–6498, ACM Press, New York, USA, July 2020.

[26] Y. Rong, W. Huang, T. Xu, and J. Huang, "DropEdge: Towards Deep Graph Convolutional Networks on Node Classification," 2020, https://arxiv.org/abs/1907.10903.

[27] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pp. 1025–1035, CA, USA, December 2017.

[28] K. Xu, C. Li, Y. Tian, T. Sonobe, K. Kawarabayashi, and S. Jegelka, "Representation learning on graphs with Jumping Knowledge networks," *In ICML*, vol. 80, pp. 5453–5462, 2018.

[29] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme, "B. P. R.: Bayesian personalized ranking from implicit feedback," pp. 452–461, 2009, https://arxiv.org/abs/1205.2618%20pp.

[30] P. K. Diederik and J. Ba, "Adam: a method for Stochastic Optimization," 2015, https://arxiv.org/abs/1412.6980.

[31] D. Liang, L. Charlin, J. McInerney, and D. M. Blei, "Modeling user Exposure in recommendation," in *Proceedings of the 25th International Conference on World Wide Web*, pp. 951–961, Québec, Montréal, Canada, April 2012.

[32] R. He and J. McAuley, "Ups and Downs: Modeling the Visual Evolution of Fashion Trends with One-Class Collaborative Filtering," in *Proceedings of the 25th International Conference on World Wide Web*, pp. 507–517, uébec, Montréal, Canada, April 2016.