Graph Neural Network for User-Item Recommendation

Zequan Xu [1] **Jiannan Gao,**[1] **Dadi Zhu,** [1] **Shuai Fan** [1] **Yuyang Li** [1]
[1] Electronic Information, School of Informatics Xiamen University

Abstract

Being a deep learning method, the application of Graph Neural Network(GNN) in recommendation systems brings about new opportunities and challenges to this domain. The architecture of GNN is well suitable for data in graph format. In this project, we dive deep into the Graph Convolutional Matrix Completion(GCMC) model and analysis its limitation in message passing step. Then we introduce the attention mechanism of Graph Attention Network(GAT) to the graph encoder of GCMC, to further utilize the information of node's neighborhood. We propose a modified model, i.e. Graph Convolutional with Graph Attention Network for Matrix Completion, GCGAT for brevity. Experiment conducted on MovieLens-100K dataset show that GCGAT outperforms the performance of GCMC in RMSE metrics. Our code is available at https://github.com/BitHub00/GCGAT.

1 Introduction

With the rapid growth of information on the Internet, recommendation systems become increasingly important for helping users alleviate information overload. The recommendation system's success makes it prevalent in many applications, including E-commerce, online advertisement and media monitoring. The core of a recommendation system is to predict how likely a user will interact with an item based on the historical interactions, e.g., click, comment, rate, browse, among other forms of interactions.

Recommendation systems collect and make use of various sources of information. It manages to retrieve a balance among factors like accuracy, disparity, and stability, which is usually referred to as the exploration or exploitation challenge(Bobadilla et al. 2013). Typical traditional methods include Collaborative Filtering and Content-based Filtering. These two methods are based on how us humans made the decision that we rely mainly on our own experience. It is widely believed that our friends' or neighbors' behavior also plays an essential role in our decision-making process.

Collaborative Filtering assumes continuity in our action. If we preferred an item, it is most likely that we will maintain this interest in the future. Given a rating profile, the algorithm locate peer users/items with a similar rating history compared to the current user/item and then generate the recommendation. From the above process, we can see that collaborative filtering algorithm does not require an understanding of the item recommended, thus makes it easy to apply in many applications.

Analogous to recommendation systems, deep learning has much impressed the world as well. The past few decades have witnessed the tremendous success of deep learning in various applications, including natural language processing and computer vision. Both academia and industry are making an effort to apply deep learning to a broader range of applications since it is capable of solving many complex tasks while providing state-of-the-art results. Therefore, the introduction of deep learning in the research of recommendation systems is an inevitable trend. In fact, it has revolutionized the recommendation architectures and overcomes obstacles faced by conventional models as well as achieving a high-quality result(Zhang et al. 2019). The non-linearity of deep learning makes it capable of capturing complicated and high-order user-item relationships.

Graph Neural Networks(GNN), a method based on deep learning that operates on graph domain, has received more and more attention recently. Due to its high interpretability and promising result, it has been widely used for graph analysis. Graph is a kind of data structure that models entities as well as their relationship, using the notation of nodes and edges, respectively. It is widely used in recommendation systems with nodes referring to user/item and edges referred to interactions. Many variants of GNN have been proposed, such as Graph Convolutional Network(GCN), Graph Attention Network(GAT), and Gated Graph Neural Network(GGNN). In this paper, we will mainly focus on the topic of GCN regarding its methodology and application, specifically in recommendation systems.

2 Related Work

In this section, we will review some related work on graph neural networks and their corresponding application in recommendation systems.

Graph Convolutional Network. Graph Convolutional Network(GCN), proposed in 2017, is a semi-supervised learning algorithm for graph-structured data(Kipf and Welling 2017). GCN extends the idea of Convolutional Neural Network(CNN)(Krizhevsky, Sutskever, and Hinton

2012) to extract features from non-euclidean structure data, which conventional CNN architecture could not handle. CNN uses kernel to calculate the weighted sum of the central pixel and adjacent pixels of a picture to form a feature map so as to achieve spatial feature extraction. The key is to determine the coefficients of the convolutional kernel by iterative optimization according to the loss function. GCN adopts spectral graph theory(Chung et al. 1997) to realize the convolution operation on graph.

More specifically, in order to represent the non-euclidean graph data, the common way is to use an adjacency matrix$\mathbf{A} \in \mathbb{R}^{n\times n}$, where n is the number of nodes in the graph, to represent the structure information in the graph. Additionally, a feature description x_i for every node i summarized in a feature matrix $\mathbf{H} \in \mathbb{R}^{n\times F}$, where F is the number of input features. The goal of GCN is to learn a function of signals/features on a graph. The model takes the adjacency matrix $\mathbf{A}$ and feature matrix $\mathbf{H}$ as input, and produces a node-level output $\mathbf{H}' \in \mathbb{R}^{n\times F'}$, where F' is the number of output features per node.

With the definition above, every neural network layer can be written as a non-linear function

$$\mathbf{H}^{(l+1)} = f(\mathbf{H}^{(l)}, \mathbf{A}), \tag{1}$$

with $\mathbf{H}^{(0)} = \mathbf{X}, \mathbf{H}^{(L)} = \mathbf{H}'$, L being the number of layers. So, the question is how to choose a proper function $f()$. In GCN, the author propose and simple but effective way of choosing function $f()$:

$$\mathbf{H}^{(l+1)} = f(\mathbf{H}^{(l)}, \mathbf{A}) = \sigma(\hat{\mathbf{D}}^{-\frac{1}{2}}\hat{\mathbf{A}}\hat{\mathbf{D}}^{-\frac{1}{2}}\mathbf{H}^{(l)}\mathbf{W}^{(l)}), \tag{2}$$

with $\hat{\mathbf{A}} = \mathbf{A} + \mathbf{I}$, where $\mathbf{I}$ is the identity matrix and $\hat{\mathbf{D}}$ is the diagonal degree matrix of $\hat{\mathbf{A}}$.The normalization step $\hat{\mathbf{A}}' = \hat{\mathbf{D}}^{-\frac{1}{2}}\hat{\mathbf{A}}\hat{\mathbf{D}}^{-\frac{1}{2}}$ is to prevent the scale of the feature vectors from being changed after the multiplication with adjacency matrix $\mathbf{A}$.

The paper uses a two-layer GCN for semi-supervised node classification and achieves quite a good result compared to baseline models.The architecture can be expressed as followed:

$$\mathbf{Z} = \text{softmax}(\hat{\mathbf{A}}'\text{ReLU}(\hat{\mathbf{A}}'\mathbf{X}\mathbf{W_0})\mathbf{W_1}), \tag{3}$$

with $\mathbf{Z} \in \mathbb{R}^{n\times c}$ being the predicted labels for nodes in the graph, where c is the class.

Despite the success in this task, deeper GCN actually has a worse performance unlike CNN where deeper architecture can result in better performance. In fact, over-fitting and over-smoothing are two main obstacles of developing deep GCN for downstream tasks. DroEdge(Rong et al. 2020) and GCNII(Chen et al. 2020) are two methods proposed to relieve these two problems.

DropEdge. DropEdge(Rong et al. 2020) is a novel and flexible technique to alleviate the over-fitting and over-smoothing issues. Over-fitting weakens the generalization ability on small dataset, while over-smoothing impedes model training by isolating output representations from the input features with the increase in network depth. The core of DropEdge is to randomly removes a certain number of edges from the input graph at each training epoch. DropEdge technique resembles to the commonly adopted Dropout technique when training deep neural network, where Dropout randomly drop units (along with their connections) from the neural network during training. The authors provide theoretical analysis as well as empirical study on servel datasets.

Simple and Deep Graph Convolutional Networks. GCNII(Chen et al. 2020) services as an extenison of the vanilla GCN model with two modifications: Initial residual connection and Identity mapping. It explains the limitation of the vanilla GCN via spectral graph theory(Wu et al. 2019a). The vanilla GCN model actually simulates a polynomial filter $(\sum_{l=0}^{K}\theta_l\mathbf{L}^l)$ of order K with fixed coefficient θ on the graph spectral domain. It's the fixed coefficient that limits the expressive power of a multi-layer GCN model and thus leads to over-smoothing. The two modifications mentioned above enable GCN to express a K order polynomial filter with arbitrary coefficients.

Inherited from the definition above, the $l-th$ layer of GCNII can be defined as

$$\mathbf{H}^{(l+1)} = \sigma\Big(\Big((1-\alpha_l)\hat{\mathbf{A}}'\mathbf{H}^{(l)}+\alpha_l\mathbf{H}^{(0)}\Big)\Big((1-\beta_l)\mathbf{I}_n+\beta_l\mathbf{W}^{(l)}\Big)\Big), \tag{4}$$

Compared to the vanilla GCN model, GCNII combines the $l-th$ representation with an initial residual connection to the first layer $\mathbf{H}^{(0)}$. This combination simulates the skip connection in ResNet(He et al. 2016). Besides, it adds an identity mapping $\mathbf{I}_n$ to the $l-th$ weight matrix $\mathbf{W}^{(l)}$. It is shown in (Hardt and Ma 2017) that a linear ResNet of the form $\mathbf{H}^{(l+1)} = \mathbf{H}^{(l)}(\mathbf{W}^{(l)} + \mathbf{I}_n)$ allows the model to put strong regularization on $\mathbf{W}^l$ to avoid over-fitting. It's more beneficial when the training data is limited.

Graph Attention Network. Graph Attention Network(GAT)(Velickovic et al. 2018) is an extension of GCN. It introduces the attention mechanism(Vaswani et al. 2017) to implicitly specify different weights to different nodes in a neighborhood without knowing the graph structure upfront. Differ from using elements of the adjacency matrix for the weighted sum of all the feature vectors of all neighboring nodes as GCN does. Therefore, it is readily applicable to inductive as well as transductive problems.

Back to GCN, the input of the model consists of adjacency matrix $\mathbf{A}$ and set of node features $(\overrightarrow{h_1}, \overrightarrow{h_2}, \ldots, \overrightarrow{h_n}), \overrightarrow{h_i} \in \mathbb{R}^F$, GCN layer will calculate a new set of features based on graph structure and characteristics of nodes, defined as $(\overrightarrow{h_1'}, \overrightarrow{h_2'}, \ldots, \overrightarrow{h_n'}), \overrightarrow{h_i'} \in \mathbb{R}^{F'}$. GCN layer first performs a feature transformation, represented by feature matrix $\mathbf{W}$, which transforms the feature vectors linearly by $\overrightarrow{g_i} = \mathbf{W}\overrightarrow{h_i}$. As for the transformed feature vector, a common way is to combine them so as to utilize the neighborhood information.

$$\overrightarrow{h_i}' = \sigma\Big(\sum_{j\in N_i}\alpha_{ij}\overrightarrow{g_j}\Big), \tag{5}$$

with α_{ij} represents the importance of node j's feature to node i's feature, or a kind of weights in other words.

The main contribution of GAT is to implicitly calculate the importance coefficient α_{ij} by attention mechanism.

Moreover, the order-preserving property of GAT explains its effectiveness. This property is proposed and proved by DeepInf(Qiu et al. 2018).

Order-preserving property. Given that $(i, j), (i, k), (i', j)$ and (i', k) represents either edges or self-loops, and $a_{ij}, a_{ik}, a_{i'j}, a_{i'k}$ are associated attention coefficients. Satisfied that if $a_{ij} > a_{ik}$ then $a_{i'j} > a_{i'k}$.

PROOF. The attention coefficient in GAT is defined as $a_{ij} = \text{softmax}(e_{ij})$, where

$$e_{ij} = \text{LeakyReLU}(p^T W h_i + q^T W h_j), \tag{6}$$

Due to the strict monotonicity of softmax and LeakyReLU, condition $a_{ij} > a_{ik}$ leads to $q^T W h_j > q^T W h_k$.

The above property indicates that, even though each node in the graph only focus on its neighbors in GAT, the calculated coefficients manage to maintain a global ranking.

Knowledge Graph Attention Network. Knowledge Graph Attention Network(KGAT)(Wang et al. 2019a) focuses on a different scenario: a hybrid structure of knowledge graph and user-item graph, where connected nodes have one or multiple linked attributes. The high-order connectivities captured by KGAT is shown to be useful for a successful recommendation. More specifically, KGAT first uses TransR(Lin et al. 2015) for the embedding of entities and relations in the graph. Then it extends the idea of GCN and GAT for recursively propagate embeddings along with high-order connectivity and reveal the importance of such connectivity;

Neural Graph Collaborative Filtering. Neural Graph Collaborative Filtering(NGCF)(Wang et al. 2019b) is a new recommendation framework that combines collaborative filtering with the neural network. The main difference lies in the embedding component, where the traditional CF model using matrix factorization to embed user/item ID as a vector(Koren, Bell, and Volinsky 2009), while the NGCF model further integrates the deep representations learned from rich side information of item(Wang, Wang, and Yeung 2015). Moreover, it replaces the traditional interaction function of inner product with non-linear neural networks.

Session-Based Graph Neural Networks. Aside from static information like ratings, dynamic information like session is also commonly seen in a recommendation system. A session can be regarded as a transaction with multiple purchased items in one shopping event(Wang, Cao, and Wang 2019). It can provide additional information on a user's short-term transactional patterns. The proposed model, called SR-GNN(Wu et al. 2019b), inherits the main idea of Long-Short Term Memory(Hochreiter and Schmidhuber 1997) and Gated Graph Neural Network. The corresponding vector for each session can be obtained through a gated graph neural network. The next step is to combine the global preference and current interest of each session with an attention mechanism for representation.

3 Proposed Solution

In this section, we will give a illustration on how to use the idea of Graph Convolutional Matrix Completion(GCMC) or user-item recommendation task in MovieLens-100K dataset. Moreover, we analysis the limitation of GCMC and propose our modified model GCGAT.

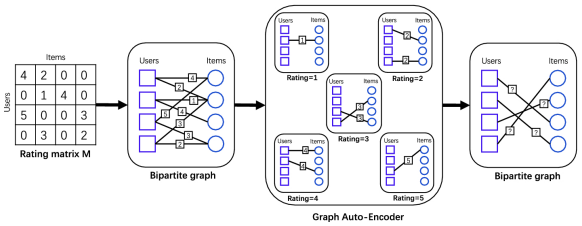

Figure 1: Notation and construction of bipartite graph

GCMC views the recommendation problem as a link prediction problem on a bipartite graph. Take dataset Movielens as example, the graph contains two types of nodes, one set consists of user nodes and the other set consists of movie nodes. There are two stages in the GCMC model. First, a graph convolutional encoder is built to embed the representation of users and movies, using both a bipartite graph and the node features. Then, the latent features are passed to a bilinear decoder, which forms the predicting matrix.

Notation

In this part, we introduce necessary notations and definitions and then formulate the problem of matrix completion in user-item recommendation task.

A generic matrix completion problem regarding the recommender systems is to fill the unknown part of the rating matrix $\mathbf{M} \in \mathbb{R}^{N_u \times N_v}$, where N_u is the number of users and N_v the number of items. The non-empty element in $\mathbf{M}$ should be a rating $r \in R$. In our experiment setting, the rating for MovieLens-100K dataset is $R = \{1, 2, 3, 4, 5\}$. The region for all known elements in the rating matrix is denoted as the set $\Omega := \{(i, j)|(M)_{i,j} \in R\}$

Constructing Bipartite Graph

The relationship between user and item can be directly indicated by a bipartite graph $G = (U \cup V, E, R)$, where $U := \{u_i\}_{i=1}^{N_u}$ is the set of user nodes, $V := \{v_i\}_{i=1}^{N_v}$ is the set of item nodes, and $E := \{(u_i, v_j, r)|u_i \in U, v_j \in V, r \in R\}$ represents the set of nodes. The weight of the edge (u_i, v_j, r) is 1 only if $(\mathbf{M})_{i,j} = r$, meaning that user u_i gives item v_j a rating of r. Therefore, for a specific rating $r \in R$, we denote the adjacency matrix as $\mathbf{M}_r \in \{0, 1\}^{N_u \times N_v}$. The demonstration of the graph construction is shown in Fig 1.

GCGAT

Our model Graph Convolution with Graph Attention Network for Matrix Completion(GCGAT) is mainly based on the work of Graph Convolutional Matrix Completion(GCMC) (van den Berg, Kipf, and Welling 2017) while introducing the key idea of Graph Attention Network(GAT) (Velickovic et al. 2018). It mainly consists of: 1) a graph encoder that embeds the user and item features by message passing in the bipartite graph; 2) a bilinear decoder that utilizes the embedded features to compute the predicted

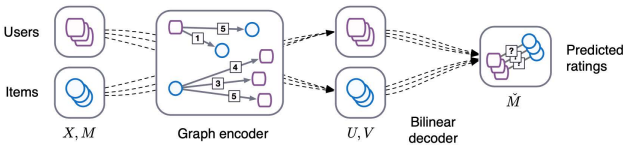

Figure 2: Illustration of model architecture

ratings. Our main contribution is introducing the attention mechanism to the message passing step of the graph encoder. The model architecture is shown in Fig 2.

Generally, a graph encoder $Z = f(X, A)$ takes a feature matrix $\mathbf{X} \in \mathbb{R}^{N \times D}$ and a adjacency matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$ as inputs, and ouputs an node embedding matrix $\mathbf{H} = [h_1, ..., h_N]^T$. The decoder predicts the missing entries in the adjacency matrix based on the pairs of node embeddings (h_i, h_j) it receives.

Graph Convolutional Encoder

The bipartite graph categories the user-item information into serveral classes regarding each specific value of rating, which allows us to encode the information through serveral channels. Suppose user i gives item j a rating of r, i.e. edge (u_i, v_j, r) is connected. Then, the message from item j to user i is formulated in the following form:

$$m_{v_j \to u_i, r} = \mathbf{W}_r^T x_j, \tag{7}$$

where x_j is the feature vector for the item j and $\mathbf{W}_r$ is the linear transformation matrix for the rating level r. This formulation of message passing aims to describe the user by using the feature from the items once they are connected in the bipartite graph. Similarly, the message can also be passed from user to items.

At the specific level of rating, we can gather the information from all the items which connect ot user i in the bipartite graph. The original operation in GCMC is to average the messages:

$$m_{u_i, r} = \frac{1}{|N_{i,r}|} \sum_{j \in N_{i,r}} m_{v_j \to u_i, r}, \tag{8}$$

with $N_{i,r}$ represents the set of neighbors of user i at the rating level r.

Once we acquire the message from each specific level of rating, we can accumulate them by stacking into a single vector representation, as proposed in GCMC:

$$h_{u_i} = \sigma(\text{concat}(\{m_{u_i, r}\}_{r \in R})), \tag{9}$$

where concat() is a matrix concatenation function and $\sigma()$ is the non-linear activation function. The acquired vector h_{u_i} can be regarded as user i's hidden feature. Analogously, the hidden representation of item j can be calculated inversely.

Limitation

However, we analysis that the strategy adopted by GCMC has some limitations. From another perspective, the averaging accumulation used by GCMC can be seen as a weighted sum of neighbor items' features, where weights for each neighbor item is assigned with the same value. It means that every item in user i's neighborhood has the same importance. Apparently this observation contradicts with the reality, where we have preference for certain item while others not.

Moreover, after acquiring the message from all rating levels, the stacking operation adopted by GCMc does not utilize the neighborhood of user or item as well. Consider the movie rating scenario, our preference for movies will be similar to people with the same ratings on the same movies. Regarding to the neighbor information of both user and item, GCMC simply recognizes which user/item belongs to the neighborhood without fully utilizing them.

Regarding to the limitation of GCMC we analysis above, we propose to introduce the idea of attention mechanism to indicate the importance of node j's features to node i.

Attention Mechanism

As we introduced, GAT implicitly specifies different weights to different nodes in a neighborhood. Here, we follow the same idea using a single-layer feedforward neural network parametrized by a weight vector $\vec{a} \in \mathbb{R}^{2F'}$, which represents the attention mechanism $a : \mathbb{R}^{F'} \times \mathbb{R}^{F'} \to \mathbb{R}$.

More specifically, we first calculate the attention coefficient e_{ij} based on node's features, take user node as example:

$$e_{ij} = a(\mathbf{W}h_{u_i}, \mathbf{W}h_{u_j}), \tag{10}$$

with weight matrix $\mathbf{W}$ being the shared linear transformation.

This formulation allows every node to attend on every other node, resulting in dropping all the structure information in the graph. Follow the modification of GAT, we inject the graph structure by only calculating e_{ij} for nodes $j \in N_i$, where N_i is node i's neighborhood. In order to fairly compare the coefficients across different nodes, a normalization step is adopted. Choosing LeakyReLU as non-linear activation function, the coefficient for measuring the importance of node j's feautres to node i is defined as:

$$\begin{aligned} \alpha_{ij} &= \text{softmax}_j(e_{ij}) = \frac{\exp(e_{ij})}{\sum_{k \in N_i} \exp(e_{ik})} \\ &= \frac{\exp(\text{LeakyReLU}(\overrightarrow{a}^T[\mathbf{W}h_{u_i} || \mathbf{W}h_{u_j}]))}{\sum_{k \in N_i} \exp(\text{LeakyReLU}(\overrightarrow{a}^T[\mathbf{W}h_{u_i} || \mathbf{W}h_{u_k}]))}, \end{aligned} \tag{11}$$

We then combine the attention coefficient with hidden representation of user/item node, to aggregate the information from its neighborhood with weights being the attention coefficient acquired:

$$h'_{u_i} = \sum_{j \in N_i} \alpha_{ij} h_{u_j}, \tag{12}$$

In this way, the hidden vector is able to encode and further utilize the imformation of its neighborhood.

To arrive at the final embedding of user node i, we transform the intermediate output h_{u_i} as follows:

$$z_{u_i} = \sigma(\mathbf{W}_\mathbf{z} h'_{u_i}), \tag{13}$$

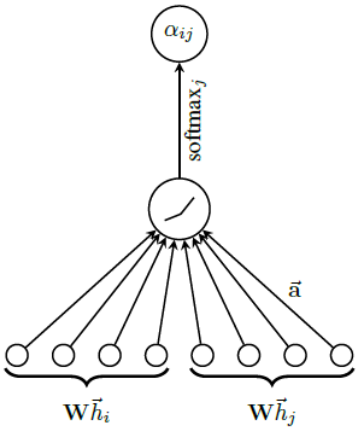

Figure 3: The attention mechanism

The item embedding z_{v_i} can be computed analogously using the same weights matrix $\mathbf{W_z}$.

Bilinear Decoder

We define the final embedding for user and item as $\mathbf{U}$ and $\mathbf{V}$ respectively. The bilinear decoder is to produce the confidence map at each level of rating. For the (i, j) position in the confidence map $\mathbf{P}_r \in [0, 1]^{N_u \times N_v}$, it should indicate the probability that the user i gives the item j a rating of r, which can be predicted by the embedded feature via a softmax function:

$$(\mathbf{P}_r)_{ij} = \frac{\exp(u_i^T \mathbf{Q}_r v_j)}{\sum_{s \in R} \exp(u_i^T \mathbf{Q}_s v_j)}, \tag{14}$$

where $\mathbf{Q}_r$ is a trainable parameter matrix, u_i^T is the ith row of $\mathbf{U}$ and v_j^T being the jth row of $\mathbf{V}$.

Utilizing this confidence map can make the rating predictions, simply as:

$$\hat{\mathbf{M}} = \sum_{r \in R} r \mathbf{P}_r, \tag{15}$$

Loss Function

During model traning, we minimize the following negative log likehood of the predicted rating matrix $\hat{\mathbf{M}}$:

$$L = -\sum_{(i,j) \in \Omega} \sum_{r=1}^{R} I[\mathbf{M}_{ij} = r] \log p(\hat{\mathbf{M}} = r), \tag{16}$$

with $I[x = y] = 1$ when $x = y$ and zero otherwise. Here, we follow the setting of GCMC to only optimize over observerd ratings.

4 Experiment

We compare our modified model GCGAT with original GCMC model. We evalute our model on a common benchmark dataset: MovieLens-100K[1]. The dataset consist of user ratings for items (such as movies) and optionally incorporate additional user/item information in the form of features. Dataset statistics are summarized in Table 1.

[1]https://grouplens.org/datasets/movielens/

Table 1: Dataset statistics

Dataset	Users	Items	Ratings	Rating Levels
ML-100K	943	1682	100,000	1,2,...,5

Table 2: Experiment Result

Model	RMSE
GCMC	0.983
GCGAT	**0.965**

In our experiment setting, we randomly split the MovieLens-100K dataset by a 4:1 ratio, into training and test data. We use RMSE as the evaluation metirc for the task of matrix completion. It can be calculated by:

$$L_{\text{RMSE}} = \sqrt{\frac{1}{|\Omega|} \sum_{(i,j) \in \Omega} \left((\hat{\mathbf{M}}_r)_{ij} - (\mathbf{M})_{ij} \right)} \tag{17}$$

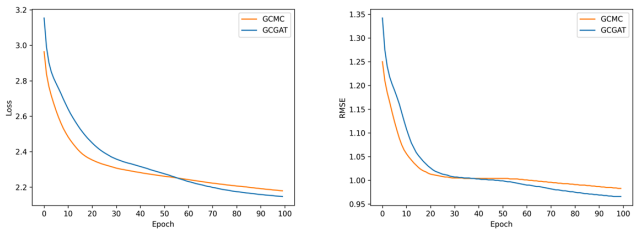

Figure 4: Loss and RMSE curve of two model

The loss curve and rmse curve during training are shown in Fig 4. The result is shown in Table 2. As indicated from the result, our idea of introducing the attention mechanism to the graph encoder works, it improves the model's performance since the model arrives at a lower value of RMSE. Lower value of RMSE indicates that the model now has a slightly better ability to predict whether and how much a user will in favor of a item.

5 Conclusion

In this project, we inherit the idea from GCMC(van den Berg, Kipf, and Welling 2017), that uses the graph convolutional encoder and the bilinear decoder to solve the matrix completion problem. Our main contribution is to introduce the attention mechanism to the messing passing step of graph encoder. The attention mechanism is used to implicitly compute the importance of node j's feature to node i's feature. We use the calculated coefficient as weights to aggregate the information of node i's neighborhood. The experiment has shown that our model can reach a lower RMSE of 0.965.

References

Bobadilla, J.; Ortega, F.; Hernando, A.; and Gutiérrez, A. 2013. Recommender systems survey. *Knowl. Based Syst.* 46: 109–132.

Chen, M.; Wei, Z.; Huang, Z.; Ding, B.; and Li, Y. 2020. Simple and Deep Graph Convolutional Networks. In *ICML*, volume 119 of *Proceedings of Machine Learning Research*, 1725–1735. PMLR.

Chung, F.; Graham, F.; on Recent Advances in Spectral Graph Theory, C. C.; (U.S.), N. S. F.; Society, A. M.; and of the Mathematical Sciences, C. B. 1997. *Spectral Graph Theory*. Conference Board of the mathematical sciences.

Hardt, M.; and Ma, T. 2017. Identity Matters in Deep Learning. In *ICLR (Poster)*. OpenReview.net.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep Residual Learning for Image Recognition. In *CVPR*, 770–778. IEEE Computer Society.

He, X.; Deng, K.; Wang, X.; Li, Y.; Zhang, Y.; and Wang, M. 2020. LightGCN: Simplifying and Powering Graph Convolution Network for Recommendation. In *SIGIR*, 639–648.

Hochreiter, S.; and Schmidhuber, J. 1997. Long Short-Term Memory. volume 9, 1735–1780.

Kipf, T. N.; and Welling, M. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *ICLR*.

Klicpera, J.; Bojchevski, A.; and Günnemann, S. 2019. Predict then Propagate: Graph Neural Networks meet Personalized PageRank. In *ICLR*. OpenReview.net.

Koren, Y.; Bell, R. M.; and Volinsky, C. 2009. Matrix Factorization Techniques for Recommender Systems. *Computer* 42(8): 30–37.

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *NIPS*, 1106–1114.

Lin, Y.; Liu, Z.; Sun, M.; Liu, Y.; and Zhu, X. 2015. Learning Entity and Relation Embeddings for Knowledge Graph Completion. In *AAAI*, 2181–2187.

Qiu, J.; Tang, J.; Ma, H.; Dong, Y.; Wang, K.; and Tang, J. 2018. DeepInf: Social Influence Prediction with Deep Learning. In *KDD*, 2110–2119. ACM.

Rong, Y.; Huang, W.; Xu, T.; and Huang, J. 2020. DropEdge: Towards Deep Graph Convolutional Networks on Node Classification. In *ICLR*. OpenReview.net.

van den Berg, R.; Kipf, T. N.; and Welling, M. 2017. Graph Convolutional Matrix Completion. *CoRR* abs/1706.02263.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, L.; and Polosukhin, I. 2017. Attention is All you Need. In *NIPS*, 5998–6008.

Velickovic, P.; Cucurull, G.; Casanova, A.; Romero, A.; Liò, P.; and Bengio, Y. 2018. Graph Attention Networks. In *ICLR*.

Wang, H.; Wang, N.; and Yeung, D. 2015. Collaborative Deep Learning for Recommender Systems. In *KDD*, 1235–1244.

Wang, S.; Cao, L.; and Wang, Y. 2019. A Survey on Session-based Recommender Systems. *CoRR* abs/1902.04864.

Wang, X.; He, X.; Cao, Y.; Liu, M.; and Chua, T. 2019a. KGAT: Knowledge Graph Attention Network for Recommendation. In *KDD*, 950–958.

Wang, X.; He, X.; Wang, M.; Feng, F.; and Chua, T. 2019b. Neural Graph Collaborative Filtering. In *SIGIR*, 165–174. ACM.

Wu, F.; Jr., A. H. S.; Zhang, T.; Fifty, C.; Yu, T.; and Weinberger, K. Q. 2019a. Simplifying Graph Convolutional Networks. In *ICML*, volume 97 of *Proceedings of Machine Learning Research*, 6861–6871.

Wu, S.; Tang, Y.; Zhu, Y.; Wang, L.; Xie, X.; and Tan, T. 2019b. Session-Based Recommendation with Graph Neural Networks. In *AAAI*, 346–353.

Zhang, S.; Yao, L.; Sun, A.; and Tay, Y. 2019. Deep Learning Based Recommender System: A Survey and New Perspectives. *ACM Comput. Surv.* 52(1): 5:1–5:38.