Group 33 — Graph-based Recommender System

Mingbo Cui, Shengzhao Lei, Futong Liu, Yuxuan Long
École Polytechnique Fédérale de Lausanne, Switzerland

***Abstract*—In this project, we first analyze our dataset and plot the user graph and movie graph. Then we use the link prediction [1] based on the bipartite graph that represents the relationship between the user and item. The message passing by graph convolution allows us to describe users using items' information, and vice versa. The rating prediction is forced to fit the user graph and item graph, such that the normalized Laplacian is used to compute the Dirichlet norms as the regularization term. The experiment has shown that our model can reach a lowest RMSE of 0.938. Even without any use of side features, the RMSE can still stay below 0.95.**

I. Introduction

With the proliferation of social media and e-commerce, recommender systems are utilized ubiquitously to predict a user's preference. The recommender system leverages the history data and aims to make personalized suggestions based on every individual user's interest. In this project, we propose to implement a recommender system as collaborative filtering from a perspective of graph.

Given that and inspired by the work of [1], we decide to view the recommendation problem as a link prediction problem on a bipartite graph: one set consists of user nodes and the other set consists of movie nodes. In short, there are two main stages. 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 our predicted rating matrix.

II. Data analysis and visualization

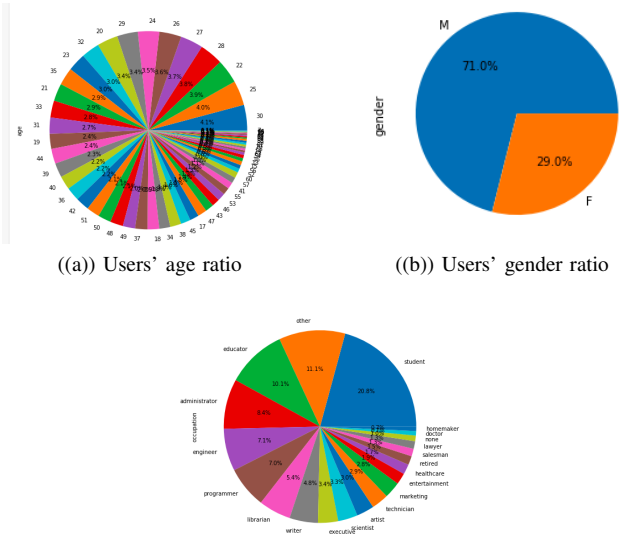
In this section, we will introduce our dataset and present some statistical information about our dataset. Besides, we also generate user graph and movie graph and do some analysis to interpret them.

A. Dataset

We use **Movielens 100k** as our dataset, which is from Movielens movie recommendation system database and contains 100,000 ratings from roughly 1000 users on 1700 movies. Various information is available about the users (e.g., age, gender, occupation, zip code) and the movies (e.g., release date, genre, movie title).

B. Statistical analysis

At first, we analysed the statistics of user data and movie data. Fig 1(a), Fig 1(b), Fig 1(c) and Fig 2 show the distributions of users' age, gender, occupation and location



((a)) Users' age ratio ((b)) Users' gender ratio

((c)) Users' occupation ratio

Figure 1: Statistics of user data

of residence respectively. The age distribution is normalized. There are some additional observations. The number of males is twice as many as that of females among users. The distribution of occupation is unbalanced since the students account for the largest partition which takes 21%. Most users reside in the US while very few users (precisely 18) reside in Canada, and in the US the distribution of users is biased towards populated areas.

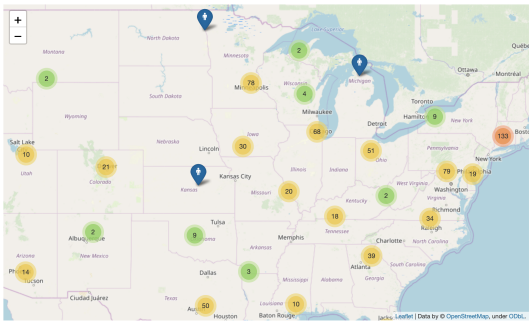


Figure 2: Distribution of user's location

As for the movie data, we found that the movies lie in a span of 77 years. Since the movies are mainly released

during 1991 and 1998, we replace the releasing year of the movies (which are indeed released before 1990) with 1990, in order to balance the distribution, as shown in Fig 3.

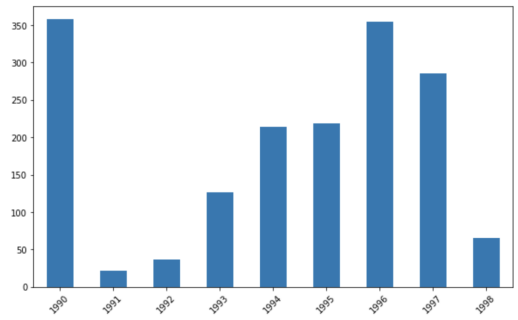


Figure 3: Distribution of Movies Across Years

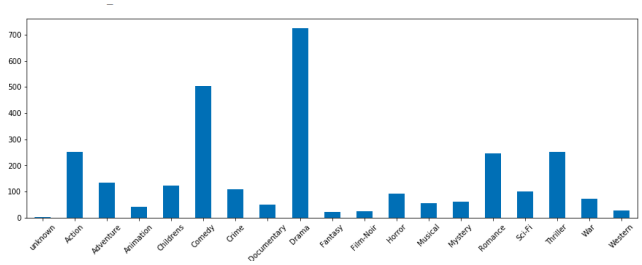
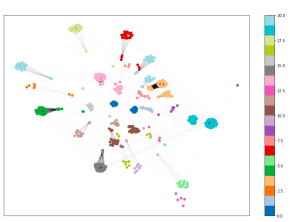
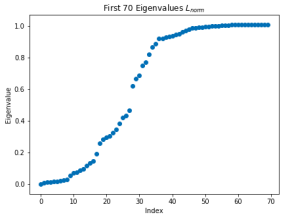


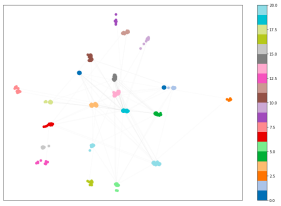
Figure 4: Distribution of Movies Genres



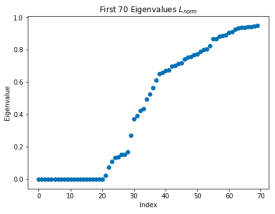
((a)) User graph I



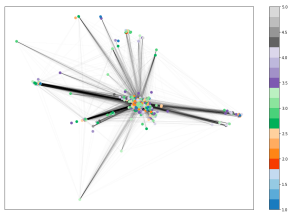
((b)) Eigenvalues



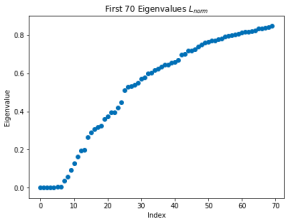
((c)) User graph II with whitening



((d)) Eigenvalues



((e)) Movie graph with whitening



((f)) Eigenvalues

Figure 5: User Graph

Furthermore, we counted the number of users ratings for each movie and assumed that this value can represent the popularity of each movie to some extent. Thus, we can get the top 5 popular movies in the dataset: Star Wars (1977), Contact (1997), Fargo (1996), Return of the Jedi (1983), Liar Liar (1997). Additionally, we showed the distribution of genres of rated movie in Fig 4 and concluded that drama and comedy are the most popular movie genres in 1997 and 1998.

C. Graph

We plotted two kinds of graph which are the user graph and the movie graph. For the user graph, we used two methods to generate the adjacency matrix: the first method (Fig 5(a)) computes the similarity of personal information of users such as age, gender, occupation and location; the second method (Fig 5(c)) is based on the previous one but with data whitening (discussed in IV-A) applied to de-correlate all the features.

As for the movie adjacency matrix, we used three methods, where the first method computes the similarity of genre between movies and the second method applies data whitening to the previous matrix. For the third method, if two movies have been rated by same user, we added one to their corresponding adjacency matrix value. Since all movie graphs are not promising, we just put the movie graph Fig 5(e) generated by the second method.

Fig 5(a) and Fig 5(c) show the constructed user graphs with colored labels representing the occupation. We find that users of the same occupation do fall into the same cluster, meaning that the occupation dominates the computation of similarity between users.

Before data whitening, from the corresponding plot of eigenvalues (Fig 5(b)), we observe that the most significant gap is at the 18th one and hence there are about 18 clusters. As we know that if the data has exactly k clear clusters, there will be a sharp gap in the Laplacian spectrum after the k-th eigenvalue. After data whitening, the eigenvalues in Fig 5(d) show a sudden 'jump' at 30 and hence 30 clear clusters. We can discover that the clusters in Fig 5(c) have clearer boundaries than those in Fig 5(a), due to the fact that the data whitening decorrelates the features of users and ease the clustering.

Fig 5(e) shows the movie graph applied with the same processing method as in the user graph, with colored labels representing the rating of a movie. Unfortunately, we cannot get any information from the movie graph. Besides, in Fig 5(f) there is no obvious jump in the plot and neither no clear cluster in the graph.

III. Model

A. Notation

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. For the known element in $\mathbf{M}$, its value should be some rating value $r \in \mathcal{R}$. For instance, in the MovieLens 100K dataset, we have $\mathcal{R} = \{1, 2, 3, 4, 5\}$. Note that the unknown element can be only assigned to a value that is not in $\mathcal{R}$. The region for all known elements in the rating matrix is denoted as the set $\Omega := \{(i, j)|(\mathbf{M})_{i,j} \in \mathcal{R}\}$.

B. Architecture

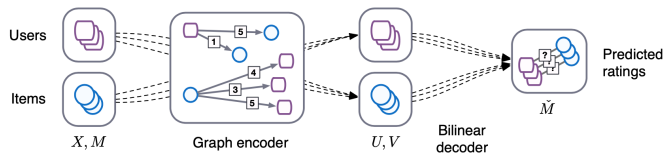


Figure 6: Simple illustration of model architecture (from [1])

Our model is based on the work in [1]. 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 ratings, as shown in Fig 6.

C. Bipartite graph

The relationship between user and item can be directly indicated by a bipartite graph $\mathcal{G} = (\mathcal{U} \cup \mathcal{V}, \mathcal{E}, \mathcal{R})$, where $\mathcal{U} := \{u_i\}_{i=1}^{N_u}$ is the set of user nodes, $\mathcal{V} := \{v_i\}_{i=1}^{N_v}$ is the set of item nodes, and $\mathcal{E} := \{(u_i, v_j, r)|u_i \in \mathcal{U}, v_i \in \mathcal{V}, r \in \mathcal{R}\}$ is the set of edges. The weight for the edge (u_i, v_j, r) is 1 only if $(\mathbf{M})_{i,j} = r$ (namely u_i gives v_j a rating of r); otherwise, the weight of the edge is assigned to 0 (namely no such rating r is given). So, for a specific rating $r \in \mathcal{R}$, we denote the adjacency matrix as $\mathbf{M}_r \in \{0, 1\}^{N_u \times N_v}$.

D. Graph convolutional encoder

This bipartite graph categorizes the user-item information into several classes regarding each specific value of rating, hence allowing us to encode the information through several channels. Suppose the user i gives the 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 [1]:

$$\mathbf{m}_{v_j \to u_i, r} = \mathbf{W}_r^T \mathbf{x}_j, \tag{1}$$

where $\mathbf{x}_j$ 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 features from the items once they are connected in the bipartite graph. Note that it can work inversely, that the message can be created from users to items.

At the certain level of rating, we can gather the information from all the items which connect to user i in the bipartite graph. This can be simply done by averaging the messages:

$$\mathbf{m}_{u_i, r} = \frac{1}{|\mathcal{N}_{i,r}|} \sum_{j \in \mathcal{N}_{i,r}} \mathbf{m}_{v_j \to u_i, r} = \frac{1}{|\mathcal{N}_{i,r}|} ((\mathbf{M}_r)_{i,:} \mathbf{X}_v \mathbf{W}_r)^T, \tag{2}$$

where $\mathbf{X}_v$ contains all features of items as its rows and $\mathcal{N}_{i,r}$ is the set of neighbours of user i at the rating level r. Note that $\mathcal{N}_{i,r}$ can be directed derived from $\mathbf{M}_r$, where one fact is that $|\mathcal{N}_{i,r}| = (\mathbf{M}_r)_{i,:}\mathbf{1}$.
Then, we can accumulate the messages from all level of rating by stacking them:

$$\mathbf{h}_{u_i} = \sigma\left(\text{concat}\left(\{\mathbf{m}_{u_i, r}\}_{r \in \mathcal{R}}\right)\right), \tag{3}$$

where concat$(\cdot)$ is a matrix concatenation function and $\sigma(\cdot)$ is the ReLU activation function [2]. Note that summing up the messages is also an alternative way, but stacking them is preferred in [1]. The ReLU function $\sigma(\cdot)$ can make the combined message sparser. Also, adding more nonlinearity can increase the representation power of the neural network. All the previous derivations for the encoder can be written into a simple matrix form:

$$\mathbf{H}_u = \sigma\left(\text{concat}\left(\{\mathbf{D}_u^{-1} \mathbf{M}_r \mathbf{X}_v \mathbf{W}_r\}_{r \in \mathcal{R}}\right)\right), \tag{4}$$

where $\mathbf{D}_u$ is a diagonal matrix with the normalization constants on its diagonal. Similarly, by sharing the same trainable parameter matrix $\mathbf{W}_r$, we can embed the items as:

$$\mathbf{H}_v = \sigma\left(\text{concat}\left(\{\mathbf{D}_v^{-1} \mathbf{M}_r^T \mathbf{X}_u \mathbf{W}_r\}_{r \in \mathcal{R}}\right)\right), \tag{5}$$

Note that the parameter sharing can alleviate overfitting. Also, the equation (4) and (5) are referred to graph convolution layer. The name 'graph convolution' comes from the fact that we convolve the features using the graph geometry.

E. One-hot feature design

At the case that the content information for users and items have been provided, we can directly inject them into the graph encoder. However, as discussed in [1], the content information may not contain enough information to distinguish different users (or items), which can lead to a severe bottleneck of informaton flow. A solution proposed in [1] is that, the input features fed into the encoder are chosen as unique one-hot vectors, i.e. $\begin{pmatrix} \mathbf{X}_u \\ \mathbf{X}_v \end{pmatrix} = \mathbf{I}$ (an identity matrix). This means that each row of $\mathbf{W}_r$ should encode the individual information corresponding to each node, at a specific rating level. To let $\mathbf{W}_r$ further realize its role in the graph encoder, here, we propose to include $\mathbf{W}_r$ when embedding the features at the encoder:

$$\bar{\mathbf{H}}_u = \text{concat}\left(\mathbf{H}_u, \{\mathbf{X}_u \mathbf{W}_r\}_{r \in \mathcal{R}}\right), \tag{6}$$

$$\bar{\mathbf{H}}_v = \text{concat}\left(\mathbf{H}_v, \{\mathbf{X}_v \mathbf{W}_r\}_{r \in \mathcal{R}}\right). \tag{7}$$

In this way, $\bar{\mathbf{H}}_u$ and $\bar{\mathbf{H}}_v$ embed both the latent information generated by message passing and the individual information encoded for each node. Note that the only trainable parameter used so far is $\mathbf{W}_r$ for each level of rating. A remark is that, even at the absence of content information, we can still perform encoding based only on the rating matrix.

F. Use of content information

The last subsection does not discuss how to use the content information if it is given for each node. As proposed in [1], the content information can be used as side features, which can be fed into a dense layer. We denote the side feature matrix for users as $\mathbf{X}_u^f$ and for items as $\mathbf{X}_v^f$. Then, continued from the equation (6) and (7), we include the transformed side features as:

$$\tilde{\mathbf{H}}_u = \text{concat}\left(\bar{\mathbf{H}}_u, \sigma(\mathbf{X}_u^f \mathbf{W}_u^f)\right), \tag{8}$$

$$\tilde{\mathbf{H}}_v = \text{concat}\left(\bar{\mathbf{H}}_v, \sigma(\mathbf{X}_v^f \mathbf{W}_v^f)\right), \tag{9}$$

where $\mathbf{W}_u^f$ and $\mathbf{W}_v^f$ are trainable matrices in the dense layer of side feature. The bias is also included in the dense layer. One more dense layer (with bias) is added to arrive at the final embedding of each user and item, simply as:

$$\mathbf{U} = \sigma(\tilde{\mathbf{H}}_u \mathbf{W}_u), \tag{10}$$

$$\mathbf{V} = \sigma(\tilde{\mathbf{H}}_v \mathbf{W}_v). \tag{11}$$

Note that the embedded features for user and item should have the same size E.

G. Bilinear decoder

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. as proposed in [1], this can be computed by the embedded feature in a softmax function:

$$(\mathbf{P}_r)_{i,j} = \frac{\exp\left(\mathbf{u}_i^T \mathbf{Q}_r \mathbf{v}_j\right)}{\sum_{s \in \mathcal{R}} \exp\left(\mathbf{u}_i^T \mathbf{Q}_s \mathbf{v}_j\right)}, \tag{12}$$

where $\mathbf{Q}_r \in \mathbb{R}^{E \times E}$ is a trainable parameter matrix, $\mathbf{u}_i^T$ is the ith row of $\mathbf{U}$ and $\mathbf{v}_j^T$ is the jth row of $\mathbf{V}$.

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

$$\hat{\mathbf{M}} = \sum_{r \in \mathcal{R}} r \mathbf{P}_r \tag{13}$$

IV. Implementation

A. Feature engineering

We apply the one-hot encoding to some integer-type feature, like movie genre and user occupation. The choice for one-hot encoding is for the favour of fairly computing distances between features.

Afterwards, we adopt ZCA [3] to perform data whitening, i.e. de-correlation. The basic idea of data whitening is to transform the data so that the covariance becomes an identity matrix. For features like $\{\mathbf{x}_i\}_{i=1}^N$, we have the covariance $\mathbf{\Sigma} = \frac{1}{N}\sum_i (\mathbf{x}_i - \mu)(\mathbf{x}_i - \mu)^T$, with the mean μ. Eigen-decomposition gives $\mathbf{\Sigma} = \mathbf{Y}\mathbf{\Lambda}\mathbf{Y}^T$. By the definition of ZCA, we have the transformation matrix:

$$\mathbf{M} = \mathbf{Y}(\mathbf{\Lambda} + \epsilon \mathbf{I})^{-\frac{1}{2}} \mathbf{Y}^T \tag{14}$$

where ϵ is a small positive number. The presence of ϵ is for the purpose of increasing numerical stability, especially when the eigenvalues are very small. Given a new sample $\mathbf{x}_{\text{new}}$, we can transform it simply as: $\mathbf{x}_{\text{clean}} = \mathbf{M}(\mathbf{x}_{\text{new}} - \mu)$. The graph Laplacian is computed using those whitened features.

B. Dropout and batch normalization

Dropout is a popular method to regularize the neural network and is also used in [1]. In this project, we replace the dropout with batch normalization [4]. Batch normalization is applied in every dense layer (from equation (8) to (11)). It can adaptively readjust the distribution of linearly transformed features before feeding into nonlinear activation.

Node dropout is also proposed in [1]. It is to randomly drop some partition of the input features from $\mathbf{X}_u$ and $\mathbf{X}_v$. This is to drop out all outgoing messages of some particular nodes. For example, we can randomly choose some rows from $\mathbf{X}_u$ and replace them with zeros. This can model the absence of data from users or items.

C. Cross-entropy and RMSE loss

Since the confidence map is the output of our matrix completion model, we can have the cross-entropy loss as in [1]:

$$\mathcal{L}_{CE} = -\frac{1}{|\Omega|} \sum_{(i,j) \in \Omega} \sum_{r \in \mathcal{R}} (\mathbf{M}_r)_{i,j} \log((\mathbf{P}_r)_{i,j}) \tag{15}$$

The RMSE is usually the evaluation metric for the task of matrix completion. It can also be a loss:

$$\mathcal{L}_{RMSE} = \sqrt{\frac{1}{|\Omega|} \sum_{(i,j) \in \Omega} \left((\hat{\mathbf{M}}_r)_{i,j} - (\mathbf{M})_{i,j}\right)^2} \tag{16}$$

In the experiment, the cross-entropy loss is found to be sufficient enough to optimize the performance. But we can also include both losses for performance guarantee.

D. Dirichlet norm regularization

The graph information used so far is presented by the graph adjacency matrix that is utilized to pass the messages between users and items. However, in our model, there is no explicit connection between our bipartite graph $\mathcal{G}$ and the predicted rating matrix $\hat{\mathbf{M}}$. To deal with it, we construct the normalized Laplacians for side features $\mathbf{X}_u^f$ and $\mathbf{X}_v^f$. Here we denote the user-Laplacian and item-Laplacian as $\mathbf{L}_u$ and $\mathbf{L}_v$ respectively. Considering one column $\mathbf{c}$ from

final predicted rating-matrix as a graph signal, it should be naturally smooth in the user graph, i.e. $\mathbf{c}^T\mathbf{L}_u\mathbf{c}$ should be as small as possible. Same for the rows of the predicted rating-matrix, hence, we compute the Dirichlet norms [5] to measure the total smoothness:

$$||\hat{\mathbf{M}}||^2_{L_u} = trace(\hat{\mathbf{M}}^T\mathbf{L}_u\hat{\mathbf{M}}) \tag{17}$$

$$||\hat{\mathbf{M}}||^2_{L_v} = trace(\hat{\mathbf{M}}\mathbf{L}_v\hat{\mathbf{M}}^T) \tag{18}$$

Therefore, we could create a loss function to regularize our predicted rating matrix, which we denote it as Laplacian loss for convenience:

$$\mathcal{L}_{Laplacian} = \frac{1}{N_u^2}||\hat{\mathbf{M}}||^2_{L_u} + \frac{1}{N_v^2}||\hat{\mathbf{M}}||^2_{L_v} \tag{19}$$

As a result, the total loss can be computed as:

$$\mathcal{L} = \lambda \cdot \mathcal{L}_{Laplacian} + (1-\lambda)\cdot(\mathcal{L}_{CE} + \mathcal{L}_{RMSE}) \tag{20}$$

where λ is the weight for this Laplacian loss.

E. Training

For faster convergence, we use full-batch gradient descent to minimize the loss. Note that mini-batch stochastic gradient descent should be used when the huge training data is given, but it is not in our case. AMSGrad optimizer [6] is chosen in this project, which is a variant of Adam [7]. Unlike Adam, AMSGrad has been theoretically proved for its convergence.

L2 regularization is also applied to all the parameters, as to reduce the possibility of overfitting. Therefore, weight decay is used during the optimization procedure.

Hyperparameters	Value
learning rate	0.01
weight decay	0.00001
number of epochs	1000
hidden layer dimension in graph convolution	5
dense layer dimension for side feature	5
final embedding dimension E	5
node dropout	0
Laplacian loss weight λ	0.05

Table I: Best hyperparameters found by grid search

V. Experiments

In the experiments, we use the hyperparameters as shown in Table I. Note that the node dropout is found to degrade the performance, so we set it to zero. We randomly split the MoiveLens 100K dataset by a 1:4 ratio, into test and training data.

Then we investigate the influence of deploying side feature, data whitening and Laplacian loss. For simple notations, let S, D and L refer to using side feature, data whitening and Laplacian loss respectively. Similarly, $\neg S$, $\neg D$ and $\neg L$ refer to no use of side feature, data whitening or Laplacian loss respectively.

As indicated from Table II, involving side feature degrades slightly the performance when Laplacian loss is not included. On the other hand, when there is a Laplacian loss component in the loss function, adding side feature will help improve the performance. This indicates the importance of imposing graph information on the prediction result. With the use of graph Laplacian loss, data whitening and side features, our model can reach a lowest RMSE of 0.938.

Ablation			RMSE
$\neg S$	D	L	0.94177
$\neg S$	$\neg D$	L	0.94147
$\neg S$	D	$\neg L$	0.94782
$\neg S$	$\neg D$	$\neg L$	0.94782
S	$\neg D$	L	0.94717
S	$\neg D$	$\neg L$	0.94481
S	D	$\neg L$	0.95684
S	D	L	**0.93799**

Table II: Ablation study to show the effectiveness of each component

VI. Future Works

The regularization using node dropout does not show its advantage in the experiment. It may be due to the fact that the adjacency matrix is already very sparse. Besides, there are two ideas proposed in [1] are very promising. First, applying weight sharing for $\mathbf{W}_r$, since each row of $\mathbf{W}_r$ is updated in different frequency. Second, regularizing on $\mathbf{Q}_r$, which is considered to be one breakthrough to our model. Furthermore, the attention mechanism [1] may replace the message averaging, so that each message can be assigned to a different level of importance. A deeper graph convolution may be desired, such that the higher order message can be also considered, instead of just using first order message in our current model.

VII. Conclusion

In this project, we borrow the idea from [1], that uses the graph convolutional encoder and the bilinear decoder to solve the matrix completion problem. There are two highlights in our works: first one is that our model can still make good rating prediction (RMSE below 0.95), even without relying on any side features; the second one is that the rating prediction is further regularized by deploying the Laplacian loss, where the graph geometry is involved. The experiment has shown that our model can reach a lowest RMSE of 0.938.

REFERENCES

[1] R. v. d. Berg, T. N. Kipf, and M. Welling, "Graph convolutional matrix completion," *arXiv preprint arXiv:1706.02263*, 2017.

[2] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, 2011, pp. 315–323.

[3] A. Kessy, A. Lewin, and K. Strimmer, "Optimal whitening and decorrelation," *The American Statistician*, vol. 72, no. 4, pp. 309–314, 2018.

[4] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[5] F. Monti, M. Bronstein, and X. Bresson, "Geometric matrix completion with recurrent multi-graph neural networks," in *Advances in Neural Information Processing Systems*, 2017, pp. 3697–3707.

[6] S. J. Reddi, S. Kale, and S. Kumar, "On the convergence of adam and beyond," *arXiv preprint arXiv:1904.09237*, 2019.

[7] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.