TECHNICAL REPORT

Course: Online Social Networks and Media

Recommender systems

Voudiotis George, RN:500
Filippou Iliana, RN:501

University of Ioannina
Department of Computer Science & Engineering

1 Introduction

In this technical report, we design and experiment with a baseline GNN approach. For this purpose, we are using the LightGCN paper [1] that is related to our project, which is going to be described in the following section. We are going to implement and experiment with the methods that are suggested and test various parameter values. Using a different dataset than the one that is being used on the LightGCN paper, we will also notice how this may affect our conclusions.

2 LightGCN Method

The LightGCN Method aims to demonstrate that existing state-of-the-art recommendation models, such as the NGCF, can be burdensome and sometimes not as effective due to their complexity. The proposed method sets as a goal the development of a lighter, yet still effective, model by performing various experiments and ablation studies on previous models, to discover the most essential parts of the recommender GCN. Based on the backbone that would be discovered, the proposed lighter model would be implemented.

2.1 Aggregation

The core of graph convolution is an aggregation function, in an attempt to learn the new representation of a target node, through the features of neighbors. The general form of this can be demonstrated as below:

$$e_u^{(k+1)} = AGG(e_u^{(k)}, \{e_i^{(k)} : i \in \mathcal{N}_u\}), \text{ k: number of layers} \tag{1}$$

More specifically, in LightGCN, the simple weighted sum aggregator is adopted and a symmetric normalisation term $\frac{1}{\sqrt{|\mathcal{N}_u|}\sqrt{|\mathcal{N}_i|}}$ is introduced to avoid a scale increase. Therefore we can define the Graph Convolution Operation as follows:

$$\begin{aligned} e_u^{(k+1)} &= \sum_{i \in \mathcal{N}_u} \frac{1}{\sqrt{|\mathcal{N}_u|}\sqrt{|\mathcal{N}_i|}} e_i^{(k)} \\ e_i^{(k+1)} &= \sum_{u \in \mathcal{N}_i} \frac{1}{\sqrt{|\mathcal{N}_i|}\sqrt{|\mathcal{N}_u|}} e_u^{(k)} \end{aligned} \tag{2}$$

2.2 Layer Combination

The only model parameters that are trainable are the 0-th layer embeddings, i.e. $e_u^{(0)}$ for all users and $e_i^{(0)}$ for all items. The embeddings at higher layers can then be computed by the above equation 2. After k-layers, we get the final representation of a user(an item) through the combination of the embeddings as follows:

$$\begin{aligned} e_u &= \sum_{k=0}^{K} a_k \, e_u^{(k)} \\ e_i &= \sum_{k=0}^{K} a_k \, e_i^{(k)} \end{aligned} \tag{3}$$

It is worth mentioning that a_k describes the importance of the k-th layer embedding and it is found that setting it to $\frac{1}{K+1}$ generally leads to good performance. Finally, regarding the model prediction, it is defined as the inner product of user and item final representations, which were described above, as follows:

$$\hat{y}_{ui} = e_u^T e_i \tag{4}$$

This is used as the ranking score for recommendation generation. Furthermore, the adjacency matrix of the user-item graph is defined as:

$$A = \begin{pmatrix} 0 & R \\ R^T & 0 \end{pmatrix} \tag{5}$$

Where $R \in \mathbb{R}^{M \times N}$, M the number of users, N the number of items and R_{ui} is 1 if there was an interaction or 0 otherwise.

2.3 Loss function

As mentioned earlier the only trainable parameters are the embeddings of the 0-th layer, $E^{(0)}$. The loss function that is being used is the Bayesian Personalized Ranking loss (BPR)[2], which is defined as follows:

$$L_{BPR} = -\sum_{u=1}^{M} \sum_{i \in \mathcal{N}_u} \sum_{j \notin \mathcal{N}_u} ln\sigma(\hat{y}_{ui} - \hat{y}_{ui}) + \lambda ||E^{(0)}||^2 \tag{6}$$

λ is the parameter that controls the regularization strength L2. As for the optimizer, Adam optimizer is being used in a mini-batch manner. It is worth mentioning that no dropout mechanisms are being used since just the L2 regularization on the embedding layer is sufficient for preventing over-fitting.

3 Implementation

For our implementation of the LightGCN method, we used the MovieLens dataset from the RecSys repository, which consists of 610 users and 9.724 movies. The movies in our case represent the items. The total number of interactions that are recorded between them is 100.836. Our aim is twofold: first, it is to predict edge existence such as user-movie selection and second, it is to predict the edge ratings that could represent for example user-movie ratings. The metrics that we used in our implementation as evaluation were the Recall@K, Prediction@K, MSE and RMSE.

$$Recall@K = \frac{\#\ relevant\ items\ in\ K}{Total\ \#\ relevant\ items} \tag{7}$$

$$Prediction@K = \frac{\#\ relevant\ items\ in\ K}{Total\ \#\ items\ in\ K} \tag{8}$$

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y})^2 \tag{9}$$

$$RMSE = \sqrt{MSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y})^2} \tag{10}$$

3.1 1^{st} Task: Predict Edge Existence

For the first task, the source represents the users and the targets are the movies. Regarding the edge attributes, we chose to only obtain the rating values that were higher than 3.0, as positive values. We then, split the dataset into a training set, a validation set and a test set, with an 80:10:10 ratio. Then, we developed the Adjacency and R matrices, followed by the operations described in the LightGCN paper method. Note that the authors dictate that regarding the loss we should use it in a mini-bath manner, so we tried mini–batches of 1024, 512 and 64 samples and concluded that 1024 leads to the best performance. For our training, we set the λ parameter of the BPR Loss at 10^{-6}. It is worth mentioning that we also experimented with different λ-parameter values in the recommended range $10^{-6}, 10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}$. In our research, we found that 10^{-6} performed best, in contrast to the Dataset used in the LightGCN paper, where it was found that 10^{-4} was optimal. Then we used the Adam optimizer, with a 10^{-3} learning rate. The total number of iterations that were performed was 5.000. For every 200 iterations, we evaluated the validation set. For validation, we used Recall@K [7], Prediction@K [8] and set $K = 20$. The training findings are presented at the Table 1 [6.1, 6.2, 6.3, 6.4].

3.2 2^{nd} Task: Rating Prediction

The only difference in our implementation for the 2nd task is that the edge attributes were rating values in the range of [0,10]. Regarding the dataset, we split the same way and used the Adam optimizer with the same learning rate, iterations and number of validation iterations. For the 2^{nd} task, we used the MSE as the loss function. For every 200 iterations, we calculated both MSE and RMSE for the validation. The results we obtained from training are shown below Table 2 [6.5, 6.6, 6.7, 6.8].

	Metrics			
#Layers	Train Loss	Val. Loss	Recall@K	Precision@K
1 Layer	−644.00244	−552.51147	0.12633	0.06269
2 Layers	−786.95953	−672.29327	0.12594	0.06124
3 Layers	−901.1792	−766.09515	0.12631	0.05937
4 Layers	−1107.27954	−853.01746	0.11738	0.05579

Table 1: Training results for 1^{st} task

.

	Metrics			
#Layers	Train MSE Loss	Train RMSE Loss	Val. MSE Loss	Val. RMSE Loss
1 Layer	2.18801	1.47919	4.97213	2.22983
2 Layers	2.18802	1.4792	4.57784	2.13959
3 Layers	2.18872	1.47943	5.57647	2.36145
4 Layers	2.19012	1.4799	5.76133	2.40028

Table 2: Training results for 2^{nd} task

4 Results

After conducting the training operation for different layers, using the test dataset, we got the following values [Table 3]. The best recall@20 is observed at layer 1, although the difference between the layers is not that significant.

	Metrics		
#Layers	Test Loss	Recall@K	Precision@K
1 Layer	−548.34351	0.09612	0.03556
2 Layers	−672.37073	0.09076	0.0353
3 Layers	−759.9126	0.09375	0.0348
4 Layers	−862.72223	0.09566	0.03691

Table 3: Test results for 1^{st} task

From Table 4 that presents the testing values, we can conclude that the best results are provided by the 4-layer model. In addition, better results are obtained with the increase in layers.

	Metrics	
#Layers	MSE	RMSE
1 Layer	4.28409	2.06981
2 Layers	4.08622	2.02144
3 Layers	5.09858	2.258
4 Layers	5.2361	2.28825

Table 4: Test results for 2^{nd} task

5 Conclusions

In this work, we aim to demonstrate in practice the main concepts and guidelines the LightGCN follows. For this, we use the MovieLens dataset to research and experiment with GCN recommendation modelling and investigate how different parameters and layers can affect the evaluation metric values, which describe the model's effectiveness. We can conclude that the proposed method was relatively easy to train, as it performs just the necessary operations, while still leading to good results. Future direction could suggest further analysis and comparison with existing similar recommendation models, to discover how each operation may affect the prediction model.

6 Appendix

6.1 Predict Edge Existance

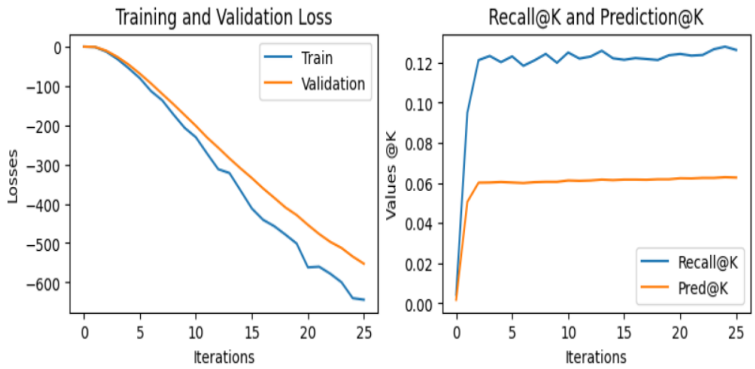

Figure 6.1: Training - Validation loss curves and Recall@K - Prediction@K at layer 1

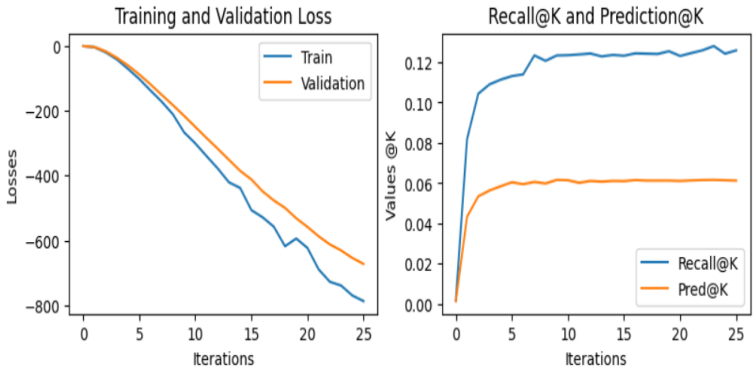

Figure 6.2: Training - Validation loss curves and Recall@K - Prediction@K at layer 2

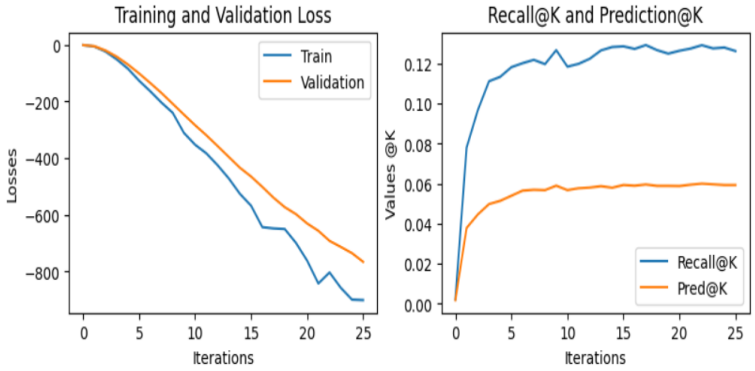

Figure 6.3: Training - Validation loss curves and Recall@K - Prediction@K at layer 3

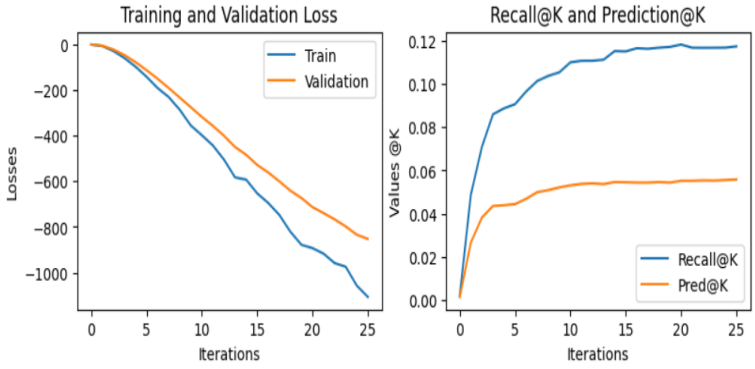

Figure 6.4: Training - Validation loss curves and Recall@K - Prediction@K at layer 4

6.2 Rating Prediction

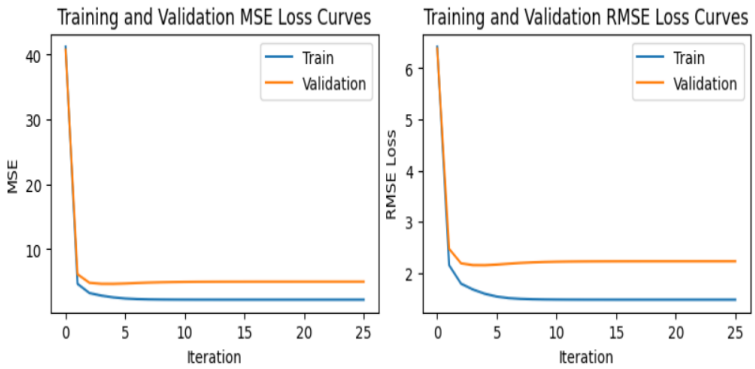

Figure 6.5: Training - Validation MSE and RMSE loss curves at layer 1

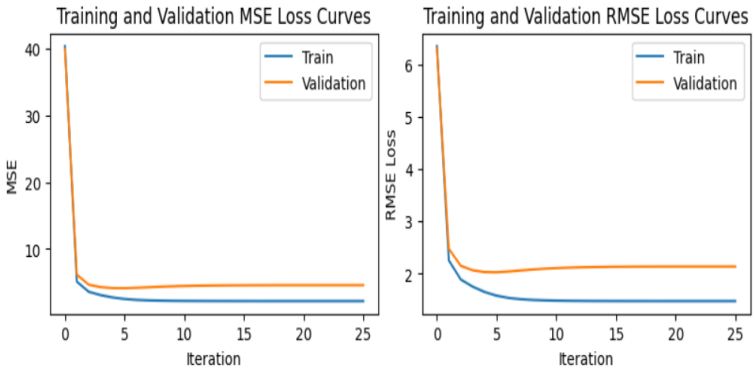

Figure 6.6: Training - Validation MSE and RMSE loss curves at layer 2

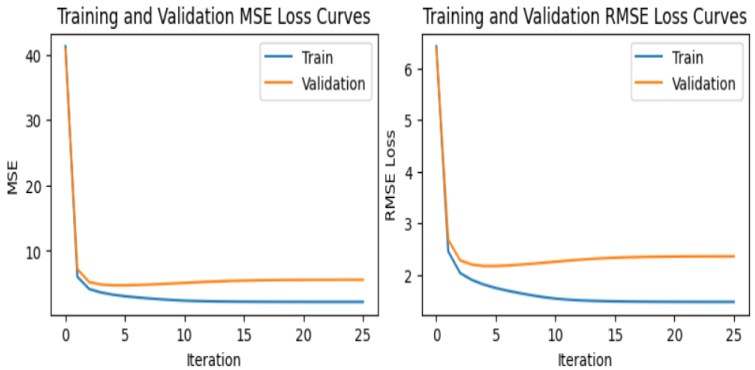

Figure 6.7: Training - Validation MSE and RMSE loss curves at layer 3

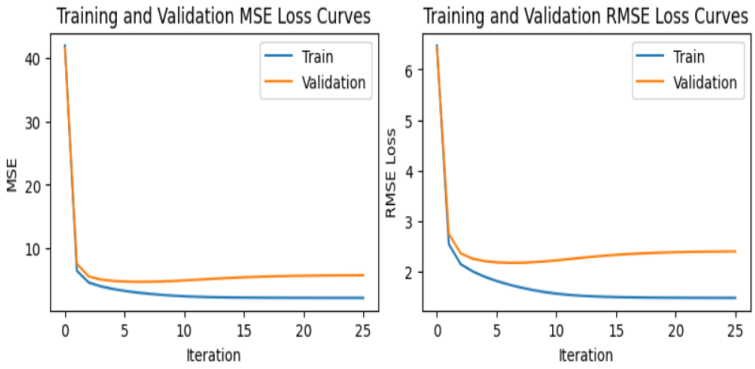

Figure 6.8: Training - Validation MSE and RMSE loss curves at layer 4

References

[1] Xiangnan He, Kuan Deng, Xiang Wang, Yan Li, Yongdong Zhang, and Meng Wang. Lightgcn: Simplifying and powering graph convolution network for recommendation, 2020.

[2] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback, 2012.