Multi-Graph Graph Attention Network for Music Recommendation

E6691.2021Spring.WADL.report

Chelsea Cui ac4788, Tong Xue tx2208
Columbia University

Abstract

In this project, we replicated a recent top-performing deep learning recommendation systems model, Mutli-Graph Graph Attention Network (MG-GAT), in PyTorch and applied it on the YahooMusic Monti dataset to get detailed performance evaluations. We chose this paper because we were interested in how it leverages attention mechanism in graph neural networks to achieve #1 ranked performance for the benchmark. We successfully replicated the model and evaluated it with the metrics provided as well as other metrics that were mentioned in the paper but not shown in results. After fine-tuning, we achieved a final Root Mean Squared Error (RMSE) result that is very similar to the result provided in the paper but slightly higher. We believe that the lack of hyperparameters and experiment setting description may cause the less accurate results. However, we also believe that our project, as the first Pytorch version of the model and the first to provide detailed evaluation results on the YahooMusic Monti dataset, may bring great reference values to the research community for future explorations.

1. Introduction

Recommendation systems have become an indispensable part of the digital space in today's world. With the exponentially increasing availability of data, numerous tech companies like Spotify, Netflix and Facebook have invested significant efforts in building and improving their recommendation algorithms to understand the user tastes and matching people with the right content [8]. Traditionally, there are two main approaches of building recommendation systems, collaborative filtering and content-based filtering [9], [10], which pose the recommendation problem as a matrix completion problem [14]. Yet, in the recent decade, with the development of deep learning techniques, researchers started to apply neural networks to recommendation systems by casting the problem into a graph-structured matrix factorization problem and utilizing geometric deep learning [4-7].

Although previous research in deep learning based recommendation systems have shown effective improvement in results compared to traditional methods, problems such as noisy information, cold-start, and high-dimensionality of data still exist. To solve these problems, in 2020, Leng et al. [2] proposed Multi-Graph Graph Attention Network (MG-GAT), which leverages sets of users' and items' information based on a neighbor importance graph. On the Papers With Code (paperswithcode.com) leaderboard, MG-GAT achieved top ranked performance for popular benchmark tasks such as Flixster Monti, Douban Monti and MovieLens 100K, and ranked as #1 for YahooMusic Monti task [11].

While MG-GAT achieved exceptional performance on the leaderboard, it is hard to understand the secret sauce behind, as the authors only mentioned limited details and only on the Yelp dataset in their paper. We hope to understand the reason why this model succeeds and how the attention mechanism makes the model different. For this reason, we decide to replicate the model in Pytorch, which has not been done before, and measure its performance on the YahooMusic Monti task. Our key objectives include: (1) reconstruct MG-GAT in Pytorch; (2) validate the experiment result for YahooMusic MontiDataset in paper (RMSE (3) obtain ranking metrics performance for YahooMusic Monti Dataset (4) perform ablation study on component importance.

Some challenges and difficulties we encountered include: (1) building a comprehensive understanding of how geometric deep learning works in recommendation systems, as neither of us have previous experience; (2) understanding the input and output dimension change of different layers and apply them in PyTorch; (3) fine-tuning the model to reproduce the result in paper; since no experiment hyperparameter detail is provided. To solve these challenges, we invested a lot of time researching and reading related surveys in the field of geometric deep learning and graph neural networks, as well as paper and code of previous works (GRAL, sRMGCNN, SVD++, GC-MC), of which our model is build upon. We believe that our project, as the first Pytorch version of the model and the first to provide detailed evaluation results on the YahooMusic Monti dataset, may bring great reference values to the research community for future explorations.

2. Summary of the Original Paper

2.1 Related Work

By leveraging the power of data, recommendation systems have shown to significantly boost the experience of users as well as performances of businesses in the recent decade [8]. Traditionally, there are two approaches to build recommendation systems: collaborative filtering and content-based filtering [9], [10]. Collaborative filtering makes recommendations based on the interests of similar users, whereas content-based filtering recommends products that are similar to the ones that users have already shown interest in.

In recent years, deep learning based algorithms have shown outstanding performance in a variety of tasks, ranging from image classification, natural language processing to speech recognition [3]. Meanwhile, researchers and companies have started applying neural networks to construct recommendation systems. Dziugaite and Roy [4] and He et al. [5] extend the matrix completion problem by using multilayer perceptron, while another popular direction is building Graph-based recommendation systems. By casting item-user relationships into geometric structures, researchers such as Rao et al. [6] and Monti et al. [7] treat the traditional matrix completion problem as deep learning on graph-structured data which aims to predict the missing links. In GRALS, He et al integrate item-item and user-user relationships as additional information to conduct graph-structured matrix factorization [6]. In RMGCNN, Monti et al. propose using spatially designed multi-graph convolution neural networks in combination with RNN to learn the underlying relationship in ratings [7]. The work we focus on in this project, however, combines the advantage of both models and introduces the attention mechanism to further improve the recommendation performance.

2.2 Methodology of the Original Paper

In the original paper, the authors introduced Mutli-Graph Graph Attention Network (MG-GAT), a module which uses sets of users' and items' information based on a neighbor importance graph [2].

The motivation for this network was to incorporate the relationship of social ties and use these different ties to build a deep learning framework. The overall purpose for this network was to promote the user's and business's local smoothness through the graph attention network (GAT) [2]. The GAT put higher weights on those neighbors who provide task-related information and lower weights on those who do not provide task-related information. This feature reduced the noisy connections in the network and meanwhile provides the relationship between different neighbors [2].

In order to explain the MG-GAT with more details, the authors defined three concepts: *Node Embedding, Edge Embedding* and *Graph Attention [2].*

$$G = (V, E) \tag{1}$$

The above equation defines a graph G which is a set of the node set V and the edge set E [2]. Node Embedding is a function which maps each node to a k-dimensional vector and edge embedding is a function which maps each edge to a k-dimensional vector [2]. Graph Attention basically means to assign weights to different nodes in a neighborhood and this weight is so called neighbor importance [2]. Moreover, the neighbor importance between user pairs and business pairs is defined as the *Neighbor Importance Graphs.* With all those in mind, the authors introduced the architecture for MG-GAT which is shown in figure 1 below.

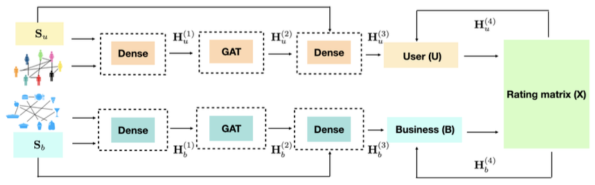


Figure 1: Overall architecture of MG-GAT [2]

The MG-GAT contains four layers, one linear dense linear, one GAT layer, one non-linear dense layer and a final aggregation layer. The dense layer takes the auxiliary information for both users and businesses as input and performance a linear transformation to produce node embeddings. Then the following GAT layer takes the output node embeddings from the dense layer as input and computes the neighbor importance. Based on the neighbor importance, the GAT layer produces the neighbor embeddings. The next non-linear dense layer takes this neighbor embedding as well as the auxiliary information at the beginning as inputs and performs a nonlinear transformation. The last aggregation layer takes the output from the previous layer and additional embeddings from the rating matrix to produce the final embeddings. Moreover, the final embeddings are used to compute the rating matrix as well as the final loss [2].

As for the model training, the authors used Adam as the optimization and the loss function L is defined as followed,

$$L = || \Omega_{training} \circ (X - \widehat{X}) ||_2^2 + \theta_1 L_{reg} \tag{2}$$

where θ_1 is a hyperparameter to control the strength of the graph regularization [2]. L_{reg} is the regularized graph regularization which is defined as followed,

$$L_{reg} = Tr(H_u^{(4)T}\hat{L}_u H_u^{(4)}) + Tr(H_b^{(4)T}\hat{L}_b H_b^{(4)}) \quad (3)$$

where $\hat{L}_u$ and $\hat{L}_b$ are the regularized Laplacian [2]; $H_u^{(4)}$ and $H_b^{(4)}$ represent the final embeddings for user and business. Meanwhile, the authors imposed L_2 regularization through on parameters such as weights and biases [2].

2.3 Key Results of the Original Paper

The authors used Yelp dataset as their main dataset where it contains users ratings and reviews on businesses. Before discussing the performance score in details, the authors introduced four evaluation metrics: *Root Mean Squared Error (RMSE), Spearman's rank-order Correlation (Spearman's correlation), Bayesian Personalized Ranking (BPR)* and *Fraction of Concordant Pairs (FCP)* [2]. The equation for RMSE is defined as followed,

$$RMSE = \sqrt{\frac{||\Omega_{test} \circ (X - \hat{X})||_2^2}{|||\Omega_{test}||_1}} \quad (4)$$

where Ω_{test} is defined as the indicator matrix. In addition to RMSE, the Spearman's correlation is also used to calculate the overall performance which is defined as the comparison between the prediction and the actual ratings. Moreover, BPR is also introduced in the evaluation which is defined as a pairwise ranking loss that is calculated from the largest posterior estimator [2]. Lastly, the authors calculated the FCP which is defined as the correct ranked pairs in the systems. The equation for FCP is defined as followed,

$$FCP = \frac{\sum_{i=1}^{N} n_c^i}{\sum_{i=1}^{N} n_c^i + \sum_{i=1}^{N} n_d^i} \quad (5)$$

where n_c^i indicates the number is corrected ranking pairs and n_d^i indicates the number of discordant pairs for user i [2].

Toke all those metrics in mind, the overall performance of MG-GAT on Yelp dataset is shown in figure 2 below,

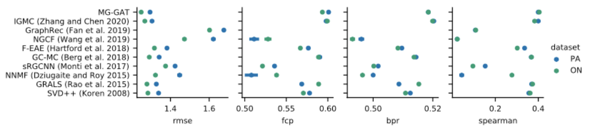


Figure 2: Performance evaluation on Yelp for RMSE, FCP, BPR and Spearman's correlation [2]

where compared with other networks such as SVD++, GRALS, NNMF, sRGCNN, GC-MC, F-EAE etc. MG-GAT performed as an effective network.

In addition to Yelp dataset, the authors also evaluated MG-GAT on four other standard datasets, MovieLens100k, Flixster, Douban and YahooMusic which are commonly used in the recommender system. The overall performance on those four datasets are shown in figure 3 below,

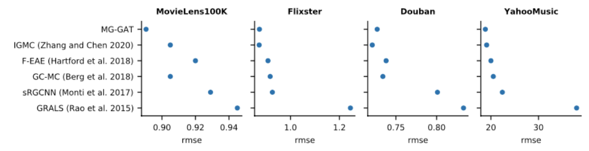


Figure 3: Performance evaluation of RMSE loss on four different data sets [2]

where the results demonstrate the robustness of MG-GAT. Among those most recent deep learning networks, i.e., GRALS, sRGCNN, GC-MC, F-EAE and IGMC, MG-GAT performs better than others on MovieLens100K and YahooMusic. Hence, in our project, we chose to reproduce this MG-GAT network with YahooMusic dataset.

3. Methodology (of the Students' Project)

In this project, we replicated the original paper in PyTorch and applied it on the YahooMusic Monti dataset. We describe the key objectives, technical challenges, engineering design and dataset details in the following sections.

3.1. Objectives and Technical Challenges

While the authors have discussed in their paper that MG-GAT has shown exceptional performance in popular benchmark tasks like Yelp restaurant, YahooMusic Monti, and MovieLens 100K recommendation, they only include model performance details on the Yelp dataset [2]. However, on the leaderboard of Papers With Code (paperswithcode.com), the model achieved the #1 ranked performance for the YahooMusic Monti task [11].

We hope to understand the how and why behind the success of this model, and thus we decide to replicate the model in Pytorch, which has not been done before, and measure its performance on the YahooMusic Monti task.

In summary, our key objectives are:

1. Reconstruct MG-GAT in Pytorch
2. Validate the experiment result for YahooMusic MontiDataset in paper (RMSE)

3. Obtain ranking metrics performance for YahooMusic Monti Dataset
4. Perform ablation study on component importance

Some technical challenges we expected and encountered are:

1. Understanding the input and output dimension change of different layers and apply them in PyTorch.
2. Fine-tuning the model to reproduce the result in paper; since no experiment hyperparameter detail is provided.

To solve the challenges, we spent a lot of time researching and reading related surveys in the field of geometric deep learning and graph neural networks, as well as paper and code of previous works (GRAL, sRMGCNN, SVD++, GC-MC). In the fine-tuning stage, we also invested time to understand the math behind the algorithm (graph-structured matrix factorization) and how each hyperparameter contributes to the final result.

3.2. Problem Formulation and Design Description

Since there is one version of TensorFlow code available on Papers With Code (paperswithcode.com), while insufficiently commented, our focus was to understand the layers and overall pipeline thoroughly, in order to be able to replicate it in PyTorch. We started by understanding the dimension change details within the model and exploring available packages in Pytorch that may help build the model. We then split the software into five major components, as shown in Figure 4, and leveraged Python code and Jupyter notebook to build the model and present results.

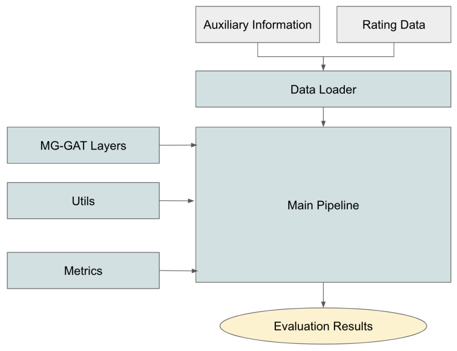


Figure 4: Software Architecture

Below is a step-by-step overview of our software architecture in pseudo code. Actions are mainly conducted in the Jupyter notebook file, which serves as the main pipeline.

Pseudo Code:

1. **data = YahooDataset(use_default_split=True)** which is defined in the datasets.py
    - It loads and splits the rating data into training and test sets
    - It also loads the auxiliary information from matlab file into graphs
2. **model = MGGAT(data, k, ,n_head,rank)** which loads the model from the MGGATlayers.py
3. Initialize criterion, optimizer, seed, epochs, loss and callback functions
4. Train model for n epoches
5. **eval_test(r_pred, r_true, data).describe()** which calculates different metrics defined in metrics.py
6. Repeat step 3-5 to fine-tune the model until results are ideal
7. Meanwhile, utils.py is called occasionally for tool functions such as **load_matlab_file(path_file, name_field)**

Since we did a thorough research to understand the original paper, we successfully replicated all functional components as described. However, we made one small technical change to improve code efficiency after researching available resources. That is, instead of reading the auxiliary information as a vector and process it manually with extensive calculation code, we utilized the dgl.graph() function provided by the Python Deep Graph Library (DGL) to convert the auxiliary information matrix directly into a graph that is suitable for our GAT layer, which is also supported by DGL.

3.3. Dataset

We evaluated our model on the YahooMusic Monti benchmark dataset [7], a subset of the KDD-Cup'11 The Yahoo! Music Dataset, which aims to challenge the community to identify user tastes through musing ratings and related information [12]. As shown in Table 1, the dataset includes 3000 users, 3000 items, and 5335 ratings. In addition, we also used the unweighted 10-nearest neighbors in the space of item features (artists, albums, and genres) provided by Monti to serve as an item graph for auxiliary information.

TABLE 1: Dataset Statistics

Users	Items	Ratings	Density	Rating Types

3000	3000	5335	0.0006	1,2..100

4. Implementation

In this section, we describe the detailed process of our implementation. We begin by describing the initial data format and how we preprocess and load the data for our PyTorch model. We then discuss how we implement the MG-GAT network in Pytorch, with high-level pseudo code.

4.1. Data Preprocessing

Data Preprocessing Github Link

The original data in this benchmark task include:

1. A .csv file of rating data which has 5335 ratings given by 3000 users to 3000 items
2. A .csv file including meta description of the dataset
3. A matlab file containing the unweighted 10-nearest neighbors in item feature space

To load data into our model, we created a data loader which splits the dataset into a training and a testing set with a 9:1 ratio, which is the common setting for models working on this task. We also provided extra flexibility to split the dataset into different train/test/validation sizes if needed. In addition, the data loader also loads the meta data as model parameters, and the matlab file as a DGL graph to serve as the auxiliary information needed by our model.

4.2. MG-GAT

The overall flow of our network is represented as figure 5 below,

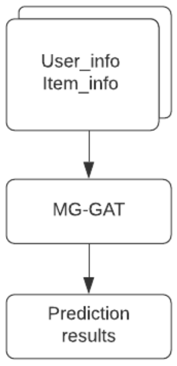


Figure 5: Overall flow of over network

where the model took in the user and item information from the data preprocessing and went through the MG-GAT to make predicted rankings. To be more specific, figure 6 below demonstrates the MG-GAT network with more details.

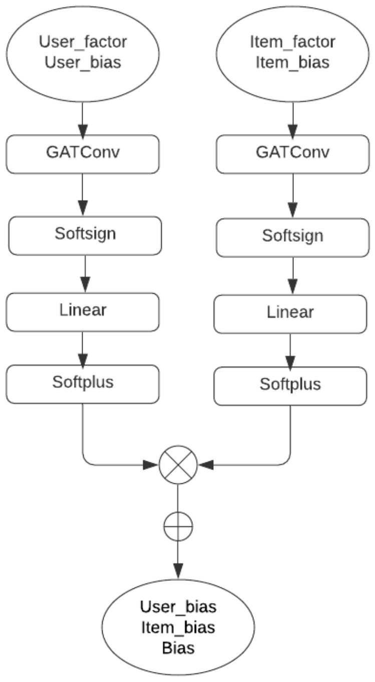


Figure 6: Detailed implementations for MG-GAT

Based on the architecture introduced by the paper, we built our MG-GAT network with multiple layers, where the user factor and bias as well as the item factor and item bias went through a GAT layer, softsign activation, linear operation and softplus activation. The results from the user side multiplied the result from the item side and added the updated user bias, item bias as well as the overall bias to get the final prediction. This operation added further flexibility to the final results.

Pseudo Code:

MG-GAT Layers Github Link

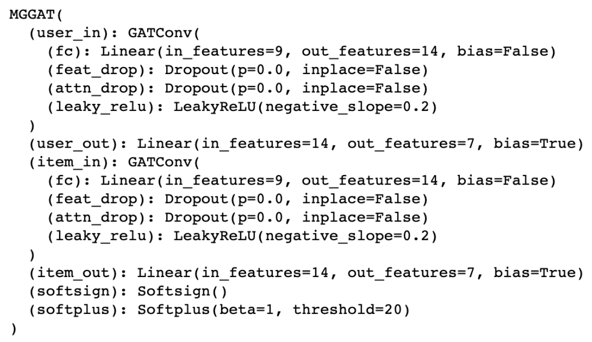
```
1. Inputs used to initialize this network: user_id,
   item_id, is_train, n_user, n_item, k, n_head, rank
2. Weights:
   a. User_factor
   b. Item_factor
   c. User_bias
   d. User_bias
3. Bias:
   a. Overall bias
4. Updated user_factor, item_factor, user_item,
   user_bias
```

5. Save n_user, n_item and resize them into user_feature, item_feature
6. Added GATConv to both user and item
7. Added Softsign function to both user and item
8. Added Linear layer to both user and item
9. Added Softplus to both user and item
10. Return = user_factor * item_factor + user_bias + item_bias + bias

5. Results

5.1. Project Results

As for the overall structure of our network, although we strictly followed the structure that was introduced by the paper, there are still a few things that are different with the original paper such as the input and output size of the features. Since these two sizes are results from the hyperparameters which are discussed in the next paragraph, they might be different with that used in the original paper. The details of our network is shown in Figure 7 below.



```
MGGAT(
  (user_in): GATConv(
    (fc): Linear(in_features=9, out_features=14, bias=False)
    (feat_drop): Dropout(p=0.0, inplace=False)
    (attn_drop): Dropout(p=0.0, inplace=False)
    (leaky_relu): LeakyReLU(negative_slope=0.2)
  )
  (user_out): Linear(in_features=14, out_features=7, bias=True)
  (item_in): GATConv(
    (fc): Linear(in_features=9, out_features=14, bias=False)
    (feat_drop): Dropout(p=0.0, inplace=False)
    (attn_drop): Dropout(p=0.0, inplace=False)
    (leaky_relu): LeakyReLU(negative_slope=0.2)
  )
  (item_out): Linear(in_features=14, out_features=7, bias=True)
  (softsign): Softsign()
  (softplus): Softplus(beta=1, threshold=20)
)
```

Figure 7 : Summary of our model

After the model training and evaluation, we finally got the RMSE loss as 22.87. From Figure 8, we could observe that the RMSE loss became stable around 23 at approximate 190 training epochs. Here we introduced three hyperparameters: *k-means, heads* and *rank*. K-means is defined as grouping items into k clusters [2]. Head is introduced in the GAT layer where it could increase the capacity of the model as well as to stabilize the learning process [13]. Lastly, rank means the low rank matrix approximation. After tuning our model, the best performance that we had is with k-means equals 9, head equals 2 and rank equals 7. In addition, the optimization used for our network is Adam stochastic optimization which is the same as the original paper. Moreover, the learning rate that is used in our network is 0.001 and the total epoch we used to train the network is 300.

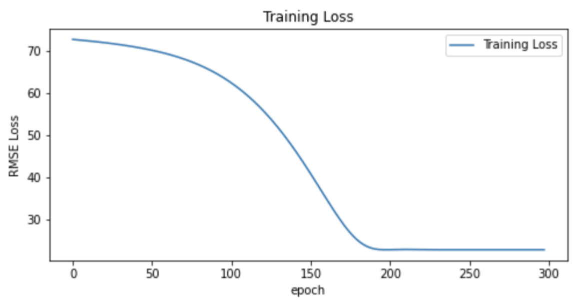


Figure 8: RMSE loss with k =9, head = 2 and rank = 7

During the process of tuning our model, we tested different combinations of the hyperparameters. The Figure 10 below demonstrates the RMSE loss across different rank values. Here we changed the rank values from 1 to 7 and the other two hyperparameters remind the same. From the graph, we could observe that the RMSE loss decreased rapidly when rank changed from 1 to 2. Moreover, the RMSE loss kept decreasing from 2 to 6. Lastly, the RMSE loss became stable when rank value is 6 and 7. Based on the result from Figure 10, our model implemented with rank value equals 7.

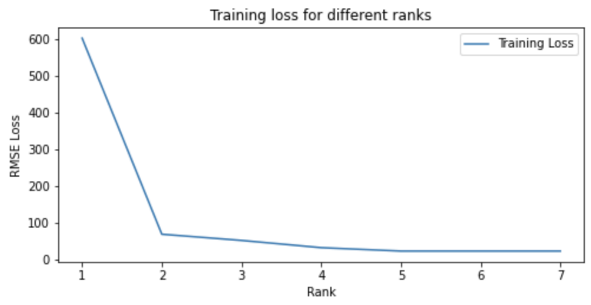
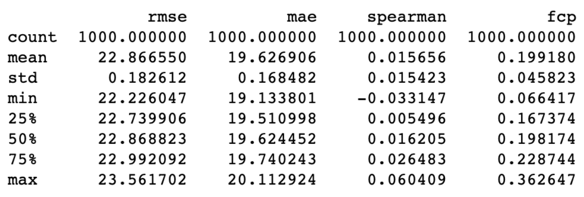


Figure 10: RMSE loss with rank from 1 to 7

In addition to RMSE, during the evaluation process, our project also included other three metrics for performance evaluation: *Mean Absolute Error (MAE), Spearman's rank-order Correlation* and *FCP.* The evaluation among these three metrics is shown in the Figure 9 below,



```
              rmse          mae     spearman          fcp
count  1000.000000  1000.000000  1000.000000  1000.000000
mean     22.866550    19.626906     0.015656     0.199180
std       0.182612     0.168482     0.015423     0.045823
min      22.226047    19.133801    -0.033147     0.066417
25%      22.739906    19.510998     0.005496     0.167374
50%      22.868823    19.624452     0.016205     0.198174
75%      22.992092    19.740243     0.026483     0.228744
max      23.561702    20.112924     0.060409     0.362647
```

Figure 9: Performance evaluation results for RMSE, MAE, Spearman's correlation and FCP

where our project displayed the mean, std, min and percentage of evaluation results. The evaluation process randomly chose 1000 pairs to calculate the metrics. As shown in Figure 9, the evaluated mean value of RMSE loss is 22.86 which is pretty close to the test result which is 22.87. This indicates that our model did performance well not only during the training process but also validated the performance during evaluation.

5.2. Comparison of the Results Between the Original Paper and Students' Project

As we reproduced the MG-GAT network with YahooMusic dataset, the original paper got a RMSE loss at 18.91 while our model got a RMSE loss at 22.87. We also compared other three metrics: MAE, Spearman's Correlation and FCP. Detailed comparison are shown in Table 2 below,

TABLE 2: Evaluation performance comparison between our model and the original paper

Model	RMSE	MAE	Spearman	FCP
Our model	22.86	19.63	0.015	0.19
Origina l paper	18.91	14.91	0.53	0.50

where we could observe that the original paper has an overall better performance than our model. We tried to improve our model by changing the data preprocessing and changing the hyperparameters, but the overall metrics still cannot be improved. The major difference between our model and the original paper is the Spearman's correlation value where the Spearman's correlation value indicates the monotonic relationship between prediction and the actual rating. The low value of the Spearman's correlation for our model demonstrates a relatively low trend for the prediction results and the actual result to change together. For some reasons causing this issue, we are thinking that there might be some difference between the training data and testing data in YahooMusic dataset. In order to improve these metrics, it might be helpful to split the training and validation dataset entirely shuffled.

The overall structure of our model and the original paper is almost the same but the paper didn't introduce some parameters they used for their optimization. The original paper only stated that the optimization they used is Adam but did not indicate the learning rate they used. In our training, we chose the learning rate as 0.001 and we did change the learning rate to other values such as 0.01 and 0.0001 but the overall losses rarely changed. As such, we think there might be other values that the original paper used is different with our model.

5.3. Discussion of Insights Gained

Once we had some issues concerning the hyperparameters, we tried to tune our model with varied hyperparameters. During the training, we changed the values of k-mean, head and rank in order to obtain a better performance. We also did some experiments such as changing only one hyperparameter and the other two hyperparameters remained the same. It turned out that the rank value changed the final RMSE loss the most. Moreover, we also tried to train our model with different epochs. Since the original paper trained their model with 1000 epochs, we started to train our model with the same number. However, it turned out that the large number of epochs incremented the overfitting problem in which the validation result is lower than the training result. Based on the above experiment, we decreased our epochs to 300 which turned out that it achieved the best performance of our model.

5.4. Ablation Study

To understand the importance of different components of MG-GAT, we also conducted an ablation study. Specifically, we investigated the importance of the GAT layer, because GAT introduces the attention mechanism and it is the major difference between MG-GAT and other models which utilizes the traditional GCN layer. By comparing the RMSE results between model with GAT and model without GAT, we found that the RMSE decreased noticeably when GAT is used. Hence, we can confirm that the use of GAT is indeed important for MG-GAT.

6. Conclusion

In this project, we successfully reproduced the MG-GAT network proposed by Y. Leng's research team with Pytorch. YahooMusic Monti benchmark dataset was used as the training and validation dataset. We evaluated our network with several metrics such as RMSE, MAE, Spearman's Correlation and FCP with respect to 22.86, 19.63, 0.015 and 0.19 respectively. In addition, our training RMSE result is 22.87 which is pretty much close to the result of the original paper but slightly higher. In order to achieve a better evaluation result, we tried to tune our model with different hyperparameter values and different sizes of epochs. We also did some other comparisons such as learning rate and different approaches for data preprocessing. During the reproduction, we realized the importance of graph and how it could be applied to improve a recommendation system. We also reveal the importance of rank with respect to the final loss.

In order to further improve our network, there are several ways that could be tried in the future. To begin with, from the dataset's perspective, since we only use the YahooMusic Monti benchmark dataset for our training and evaluation, the training results could be improved by applying the model to the entire YahooMusic dataset. The YahooMusic Monti benchmark dataset contains 5335 rating values from 3000 users and 3000 items which is only a portion of the original dataset. In the future, training the model with the entire dataset might be able to improve the performance of our model. Moreover, as for the side of our model, it could be improved by tuning with different values of the three hyperparameters. We have tried our best to tune the model in this project, but we also noticed the effectiveness of changing the rank value with respect to the improvement of the final performance. As such further improvements on hyperparameters might be able to improve the overall performance of our model.

7. Acknowledgement

We would like to thank Professor Zoran Kostic and our TA Zihao Xiong for designing this course, and arranging the assignment and project to guide us dive deep into the field of deep learning. We could also thank our fellow classmates for creating the lectures on different topics which inspired us to work on this project.

8. References

[1] https://github.com/acui34/mg-gat-pytorch
[2] Y. Leng, R. Ruiz, X. Dong, A. Pentland, "Interpretable Recommender System with Heterogeneous Information: A Geometric Deep Learning Perspective", Sep. 2020
[3] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang and P. S. Yu, "A Comprehensive Survey on Graph Neural Networks," in IEEE Transactions on Neural Networks and Learning Systems, vol. 32, no. 1, pp. 4-24, Jan. 2021
[4] GK. Dziugaite, DM. Roy (2015) Neural network matrix factorization. arXiv preprint arXiv:1511.06443
.[5] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, TS. Chua (2017) Neural collaborative filtering. Proceedings of the 26th
international conference on world wide web, 173–182.
[6] N. Chua, HF. Yu, PK. Ravikumar, IS. Dhillon (2015b) Collaborative filtering with graph information: Consistency and scalable methods. Advances in neural information processing systems, 2107–2115.
[7] F. Monti, BM. Michael, and B. Xavier, Geometric matrix completion with recurrent multi-graph neural networks." arXiv preprint arXiv:1704.06803 (2017).
[8] D. Jannach, P. Resnick, A. Tuzhilin, M. Zanker, (2016) Recommender systems beyond matrix completion. Communications of the ACM 59(11):94–102.
[9] J. Breese, D. Heckerman, and C. Kadie, Empirical Analysis of Predictive Algorithms for Collaborative Filtering. In Proc. Uncertainty in Artificial Intelligence, 1998.
[10] M. Pazzani, and D. Billsus, Content-based Recommendation Systems. The Adaptive Web, pp. 325–341, 2007.
[11] "Papers with Code - Interpretable Recommender System With Heterogeneous Information: A Geometric Deep Learning Perspective," Interpretable Recommender System With Heterogeneous Information: A Geometric Deep Learning Perspective | Papers With Code. [Online]. Available: https://paperswithcode.com/paper/interpretable-recommender-system-with. [Accessed: 24-Apr-2021].
[12] Dror, Gideon, et al. "The yahoo! music dataset and kdd-cup'11." Proceedings of KDD Cup 2011. PMLR, 2012.
[13] H. Zhang, M. Li, M. Wang, & Z. Zhang. "Graph attention network". Retrieved April 25, 2021, from https://docs.dgl.ai/en/0.4.x/tutorials/models/1_gnn/9_gat.html
[14] E. Candes, and B. Recht, Exact Matrix Completion via `Convex Optimization. Foundations of Computational Mathematics, 9(6):717–772, 2009.

9. Appendix

9.1 Individual Student Contributions in Fractions

	ax4788	tx2208
Last Name	Cui	Xue
Fraction of (useful) total contribution	1/2	1/2
What I did 1	Coding	Coding
What I did 2	Training Model	Training Model
What I did 3	Writing Report	Writing Report