LightGCN Recommender System on Goodreads Dataset

Authors
Jason Nguyen, Christopher Garrett
Rice University
jn61@rice.edu, cg100@rice.edu

Abstract

Recommender systems are one of the most widely utilized applications of graph machine learning. From a bipartite graph of users and items, a recommender system seeks to predict what items best fits to each user based on the interaction data and similarities within the graph. We discuss and explore the effectiveness of two state of the art recommender system architectures, matrix factorization and LightGCN. We evaluate the performance and compare of both models on different size subsets of the GoodReads interactions and books dataset, and see the effectiveness of LightGCN over standard matrix factorization. Both of our implementations were done in Pytorch.

1 Introduction

Recommender systems are used to provide personalized recommendations across a wide array of domains, from social networks to product recommendations. Standard recommender systems involve representing the relationship between users and items as a bipartite graph, where edges represent interactions such as ratings, purchases, or views. Recommender system then utilize similarities between users who share similar interactions between items to generate recommendations.

Matrix factorization is a staple technique for recommender systems; it decomposes the interaction matrix into lower-dimensional components in an attempt to learn latent characteristics of the relationship. The dot product of the user embeddings and item embeddings will ideally converge towards the true interaction between them. For our case, this would be the user rating from 1 to 5 of a book from GoodReads. This method is fast and interpretable, but does not scale well and tends to struggle achieving high performance.

Hence, researchers are exploring graph-based machine learning approaches, such as LightGCN (Light Graph Convolutional Network). Normal GCNs try to learn node representations by smoothing over graph features via iterative graph convolution. In other words, GCNs aggregate features of the target node's neighbors to represent the the target. LightGCN in particular is well optimized for bipartite graph recommender systems, since it does not utilize the target node itself in aggregating a node representation. It heavily simplifies traditional GCN methods by only utilizing the node's neighbors. After aggregating and combining all the layers of an embedding, a prediction is made by the inner production of the final user and item embeddings.

In this report, we explore the performance of matrix factorization and LightGCN on varying sized subsets of the GoodReads interactions and books dataset through PyTorch. We seek to measure and compare the accuracy and scalability of both methods.

Final Project for Machine Learning with Graphs.

2 Models

2.1 Matrix Factorization

The matrix factorization model initializes embeddings randomly and uniformly for each unique user and item node we have in our interaction graph. Matrix factorization functions on the premise of collaborative filtering, where recommendations are generated by filtering out items a user may like based on other users who have liked similar items. In other words, recommendations are made by looking at users with a similar set of interactions to the same distribution of items. The model then learns by calculating the dot product of the user embeddings and the item embeddings to be the interaction score and trying to minimize the difference between the prediction and the true interaction score. The final embeddings should be able to replicate the original scores.

2.2 LightGCN

Initial user and item embeddings are similarly generated randomly for each node. LightGCN then performs graph convolution with the following propagation rule [2]:

$$e_u^{(k+1)} = \sum_{i \in N_u} \frac{1}{\sqrt{|N_u|}\sqrt{|N_i|}} e_i^{(k)}$$
$$e_i^{(k+1)} = \sum_{i \in N_i} \frac{1}{\sqrt{|N_i|}\sqrt{|N_u|}} e_u^{(k)}$$

for $e_i^{(k)}$ and $e_u^{(k)}$ the item and user embeddings at the kth layer. $|N_u|$ and $|N_i|$ are the number of user and item neighbors of the current node that the target is being propagated on.

Note that the propagation rule does not require the target node, only the connected neighbors of the node it is being propagated with. Self connections are not needed in comparison to normal GCN methods.

Then, the final user and item embeddings are produced by summing up the weighted embeddings at each layer:

$$e_u = \sum_{k=0}^{K} \alpha_k e_u^{(k)}$$
$$e_i = \sum_{k=0}^{K} \alpha_k e_i^{(k)}$$

for $\alpha_k \geq 0$ a weighted hyperparameter to tune. This combination of layers prevents over smoothing of the final embeddings (preventing loss of informaiton and over simplicity), better represents the information between the interactions, and replicates the effect of having a self connection in the propagation rule. Predictions are then made by the inner product of the final representations for a user and item:

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

where a higher ranking score indicates a better recommendation. Besides the input parameters and inital layer embeddings, LightGCN has no other trainable parameters which also greatly improves its speed over traditional GCN methods.

3 Experiments

3.1 Setup

Our dataset is from late 2017 Goodreads [4] users who made their shelves public; anyone can freely access their ratings and reading history without a login. Since the dataset is quite large, we utilize only user-item interactions involving books within the poetry genre. This subset has 36,514 unique books, 377,799 unique users, and 2,734,350 interactions. We further filter out for only positive reviews (ratings greater than or equal to 3) to ensure our models focus on only generating the best positive recommendations possible. Furthermore, we keep only users that have reviewed 10 or more books and books that have appeared 3 or more times to prevent sparsity. Our final dataset which we drew samples from had 20,858 unique books, 23,755 unique users, and 554,013 interactions.

We created five different samples with 100, 500, 1000, 3000, and 5000 unique users. Our training and testing sets were then created by stratifying on unique user ids to ensure all users would appear in both sets. For each user, we removed 20 percent of their interactions to save for our test set.

3.2 Training

We train over 40 epochs with batch sizes of 1024. The Adam optimizer was used for both models. Matrix factorization had a learning rate and weight decay of 1e-4, while LightGCN had a learning rate of 5e-3 and weight decay of 1e-3. For loss, we utilized bayesian personalized ranking (BPR) loss [3] and negative samplings. Negative interactions were defined as user-item pairs with books the user has not rated, or equivalently has rated less than 3 stars.

$$\text{BPR Loss} = \log(1 + \exp(\text{negative scores} - \text{positive scores}))$$

where the positive and negative scores are the predictions generated by our model for a positive user-item pair and negative user-item pair respectively. We want to minimize this difference under the softplus activation function, which is similar to ReLU but constraints our output to be positive.

To evaluate, both models will produce k = 10 recommendations that will be compared to the true interactions in the test set. For evaluation metrics, we use recall@k, precision@k, normalized discounted cumulative gain (NDCG@k) for k predictions, and mean average precision (MAP@k) for k predictions. NDCG measures the quality of recommendation rankings, where the position is important. Indeed, recommendations that appear earlier would ideally be more relevant to the user. MAP summarizes the recall and precision when working across multiple recommendations, given an overall evaluation of the model. All of these metrics range from 0 to 1, where 1 indicates perfect rankings or results and 0 means terrible performance with no relevant results.

4 Results

We present the final training and validaiton results for each model:

Table 1: Results for Matrix Factorization, K = 10

#Unique Users	**#Interactions**	**Loss**	**Recall**	**Precision**	**NDCG**	**MAP**
100	2711	0.6420	0.2269	0.0482	0.1488	0.0967
500	12386	0.4697	0.1528	0.0442	0.1071	0.0627
1000	24152	0.4313	0.1271	0.0365	0.0947	0.0574
3000	71367	0.4418	0.1278	0.0382	0.0962	0.0573
5000	116541	0.4743	0.1267	0.0388	0.0955	0.0561

Table 2: Results for LightGCN, K = 10

#Unique Users	#Interactions	Loss	Recall	Precision	NDCG	MAP
100	2711	0.0354	0.1526	0.0552	0.1275	0.0780
500	12386	0.0394	0.1828	0.0676	0.1603	0.0983
1000	24152	0.0364	0.1921	0.0683	0.1635	0.0994
3000	71367	0.0391	0.2014	0.0709	0.1705	0.1048
5000	116541	0.0397	0.2056	0.0722	0.1724	0.1048

Matrix factorization provided better results for the smallest subsampled data, but had increasingly worse performance as the number of interactions increased. Furthermore, despite several different learning rates and optimzers such as stochastic gradient descent (SGD) and adaptive gradient algorithm (Adagrad) being tested, the model would still failed to converge to a loss less than 0.1. The training time for 5000 unique users was 1280.62 seconds.

LightGCN scaled significantly better with more interactions introduced, demonstrating consistent increasing performance and better scalability than matrix factorization. The training time for 5000 unique users was 1023.32 seconds, being several minutes faster as well.

While precision remains low for both methods, this is mainly due to the choice of k = 10 and the number of true interactions between users varying significantly. Some users had hundreds of interactions while others had the bare minimum of 10 that we required. Since we wanted to focus on capturing positive trends and recommendations, a high recall shows that we were able to capture more of the relevant interactions. We deem false positives in our precision low stakes, as long as we are able to correctly a few relevant items.

With further tuning, more epochs, and a larger quantity of data, we suspect LightGCN may yield even better performance. Indeed, below we show the training plots for both methods and see that LightGCN still demonstrates a slight upward trend in performance and decrease in loss, even near the end of training.

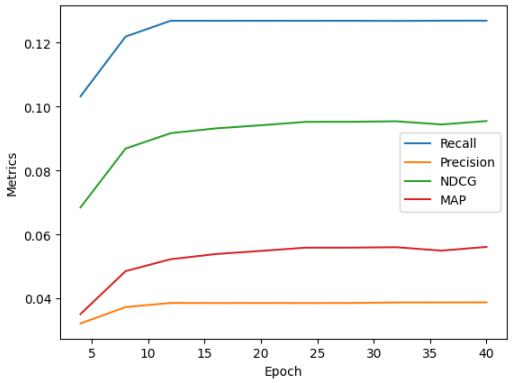

Figure 1: Matrix factorization

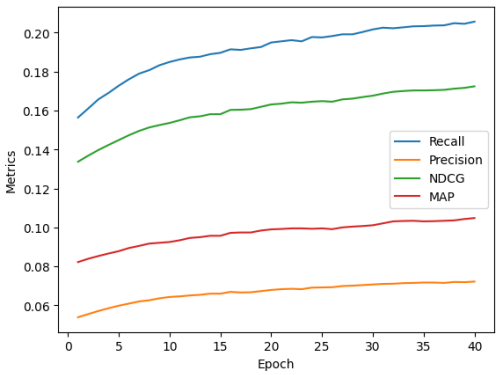

Figure 2: LightGCN

Figure 3: Validation metrics for 5000 unique users

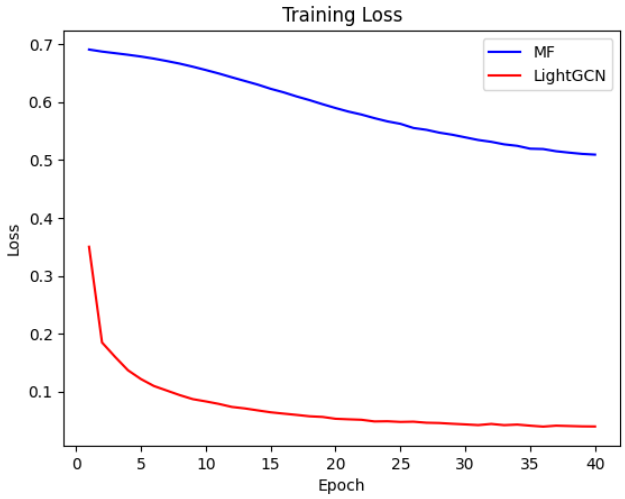

Figure 4: Training losses for 5000 unique users

The benchmarks shown in the work pioneering LightGCN [2] utilized the Gowalla, Yelp2018, and Amazon-Book datasets. We show that our work has demonstrated comparable results for k = 10. We do concede that our dataset is smaller in quantity and that using only interactions in the poetry subgenre may have influenced our results in comparison to these more diverse datasets .

Table 3: Our LightGCN results vs other dataset benchmarks [2]

Metric	**Goodreads**	**Gowalla**	**Yelp2018**	**Amazon-Book**
Recall	0.2056	0.1830	0.0649	0.0411
NDCG	0.1724	0.1554	0.0530	0.0315

5 Challenges

There were three main challenges when performing our experiments: dataset splitting, choice of loss, and setting up the LightGCN architecture.

We explored two main ways to split our dataset into a train and test set; setting random ratings to 0 or removing a portion of interactions for each user. The first method allows us to ensure that all users and items will appear in our train set and prevent the models from needing to predict on unseen items in the test set. Since ratings are from 1 to 5, a rating of 0 will imply that the user has not read or rated that book yet. This would be the equivalent of 'hiding' an interaction and seeing if the model can correct predict it. However, we did not want to introduce a negative bias into our model, since a 0 may affect the calculations of the inner products when training.

Instead, we opted to remove a portion of interactions for each user for our test, which would ensure all users appeared in both sets of data. The weakness of this was that sometimes an item would appear in our test set but not our training set if the item was sparse. We tried to filter out for sparse items but this was still difficult to avoid. However, we felt this method of data splitting was still better appropriate for training.

The original choice of loss was MSE, since that's the general and most interpretable loss when doing matrix factorization. However, we noticed that MSE resulted in very poor or stagnating improvements in NDCG and MAP when validating. MSE is also typically not applied with LightGCN, which was originally proposed with BPR loss. Instead of applying MSE to LightGCN, we explored applying

BPR to matrix factorization. We had to rewrite our entire matrix factorization training loop and preprocessing to accommodate this, but it provided much better results. However, we do concede that our matrix factorization training loop could have still been more sophisticated and may have been able to provide slightly better results. We wanted to get a loss less than 0.1 but failed.

Finally, for setting up LightGCN, there is the base module from the PyTorch geometric library that we could have utilized. However, we wanted more flexibility and to better understand how it functions. In particular, we wanted an implementation that could use data with minimal preprocessing and directly from an interaction dataframe. Hence, we explored several difference scratch implementations in PyTorch [1] [5]. Since our final implementation had results comparable to the original paper, we felt that we succeeded.

6 Discussion

We originally wanted to utilize more interaction features, such as review text length, that could have provided more insight into what users may have enjoyed. We wanted to propose that a longer positive review text length would typically be higher engagement and enjoyment, and would have wanted to put more weight on those interactions. However, we were unable to implement this effectively. Other features we wanted to include were node features, such as number of total books reviewed for a user and the author name for a book. Node features would have allowed us to have done more than a random initialization of node embeddings and provided a better start in training. Regardless, this was still a great exercise into graph machine learning and recommender systems.

7 Conclusion

In this work, we explored the architechtures of matrix factorization and LightGCNs and evaluated their effectiveness on varying sized datasets from Goodreads. Our conducted experiments demonstrated that LightGCN is superior in performance and scalability to matrix factorization and has further room for improvement. We believe LightGCN is a strong standard for recommender systems going forward, due to its interpretability and efficiency compared to traditional GCN frameworks.

8 Hyperlinks

Github link to sourcecode

References

[1] D. Das. Lightgcn->pytorch(from scratch).

[2] X. He, K. Deng, X. Wang, Y. Li, Y. Zhang, and M. Wang. Lightgcn: Simplifying and powering graph convolution network for recommendation, 2020.

[3] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback, 2012.

[4] M. Wan and J. J. McAuley. Item recommendation on monotonic behavior chains. In S. Pera, M. D. Ekstrand, X. Amatriain, and J. O'Donovan, editors, *Proceedings of the 12th ACM Conference on Recommender Systems, RecSys 2018, Vancouver, BC, Canada, October 2-7, 2018*, pages 86–94. ACM, 2018.

[5] A. Zhou. Lightgcn with pytorch geometric.