COMPARING GRAPH-BASED COLLABORATIVE FILTERING TO DEEP CONTENT-BASED APPROACHES FOR MOVIE RECOMMENDATIONS

AALOK PATWA (APATWA@SEAS)

ABSTRACT. I implement, train, and evaluate two deep learning architectures for recommendation systems: LightGCN, a graph convolutional approach for collaborative filtering, and DLRM, a neural network for content-based filtering. I observe that LightGCN outperforms DLRM on MovieLens-1M, a dense movie ratings dataset, but that its performance degrades on a sparser dataset, Amazon Fashion. I demonstrate that DLRM shows potential as a content-based recommender despite its poor performance on MovieLens, but it requires large amounts of compute in order to make recommendations on the test set. I also show that LightGCN performs reasonably well regardless of the embedding dimension, pointing to its expressive power given its simplicity. Ultimately, I provide a perspective on the strengths and weakness of graph neural networks for collaborative filtering versus deep learning for content-based recommendation.

1. INTRODUCTION

1.1. **Background.** The rise of online platforms has created tremendous demand for recommendation systems, a class of machine learning algorithms designed to predict users' preferences for specific items. These systems play an important role in the operations of websites like Amazon, Facebook, and Netflix, where users are recommended products, posts, and media, respectively [1]. Quality recommendation is essential, since users will only stay on the platform if they see content they are interested in. The proof for this is in the pudding: some estimates show that 75% of the content viewed on Netflix came from recommendations [2].

There are two main recommendation system paradigms: collaborative filtering and content-based filtering [3]. In collaborative filtering, we make predictions for a user based on the preferences of similar users. Content-based filtering, by contrast, involves predicting whether a user will like a certain item based on explicit features of the user and the item being assessed. My goal in this work is to gain familiarity with popular deep learning architectures for both collaborative filtering and content-based filtering and gain intuition into their strengths and weaknesses.

1.2. **Contributions.** In this work, I compare and contrast the performance of a graph convolutional neural network for collaborative filtering and a deep neural network for content-based filtering. I first create two from-scratch PyTorch implementations of prominent recommendation system architectures: Light Graph Convolutional Network (LightGCN) and Deep Learning Recommendation Model (DLRM). Then, I evaluate the performance of these systems on the MovieLens-1M dataset, a popular recommendation system benchmark dataset. I also create a novel benchmark for LightGCN on a sparser dataset, Amazon Fashion Reviews, which confirms that sparsity degrades the performance of collaborative filtering methods. Finally, I discuss the results of varying LightGCN's embedding dimension.

1.3. **Related Work.** Graph convolutional neural networks have emerged as one of the best-performing architectures for collaborative filtering [4]. In this work, I study LightGCN, a graph convolutional network created in 2020 [5]. The authors showed that LightGCN outperformed NGCF, another prominent graph-based collaborative filtering architecture, on three different recommendation system datasets. However, LightGCN has not been extensively tested on the MovieLens-1M dataset. Some work has evaluated a modified LightGCN architecture on MovieLens [6], but there still does not exist a solid benchmark for the original model.

Meta open-sourced DLRM as a model to predict click-through rate for content on their social media platforms [7]. However, to my knowledge, DLRM has not been evaluated on the MovieLens dataset, and there is not much literature analyzing its performance on applications other than clickthrough rate.

2. APPROACH

2.1. **Datasets.** MovieLens-1M is a tabular dataset containing 1,000,209 ratings of 3,882 movies by 6,040 users [8]. Each rating is a score from 1-5. The dataset also contains the title and relevant genres (as strings) for each movie, along with the age, gender, occupation, and zipcode for each user. I used the MovieLens-1M dataset as the primary dataset to evaluate both LightGCN and DLRM.

In addition, I use the Amazon Fashion reviews [9] data to benchmark the performance of LightGCN on a sparser user-item graph. This dataset contains 883,636 total ratings from 1-5, but with 749,233 different users and 186,189 different products. This means that the average user only has about 1.18 ratings in the dataset, making recommendation difficult.

2.2. **LightGCN.** LightGCN is a graph convolutional neural network designed to learn an embedding, or a fixed-length vector representation, of each user and item that we want to recommend (Figure 1).

It models relationships between users and items as a bipartite graph, with "user nodes" and "item nodes". There is an undirected edge from user u to item i if rating $r(u, i)$ is "positive," with the definition of "positive" being context-dependent. In this work, we defined a positive interaction to be a rating of 5/5. So, the graph shows the relationships between users and the items that they have rated 5/5. This graph is instantiated on the users and items present on the training set.

Given a certain user-item graph, the first layer of LightGCN gives each user and item node in the graph a learnable embedding. The subsequent layers are non-parametric graph convolution layers. Each layer sets the embedding for each node to a weighted sum of its neighbors' embeddings. Therefore, the embedding for a user u becomes a weighted sum of the embeddings of the items in its neighborhood. Specifically:

$$e_u^{l+1} = \sum_{i \in N(u)} \frac{1}{\sqrt{|N(u)|}\sqrt{|N(i)|}} e_i^l$$

$$e_i^{l+1} = \sum_{u \in N(i)} \frac{1}{\sqrt{|N(i)|}\sqrt{|N(u)|}} e_u^l$$

After each graph convolutional layer, we store the resulting embedding. At the end of the network, the final output embedding for each node is the mean of that node's embedding at each layer.

The score of test user embedding u_t and test item embedding i_t is $e_u \cdot e_i$. To make recommendations for user u, we calculate its score with all items it has not yet rated, and recommend the top-scoring items.

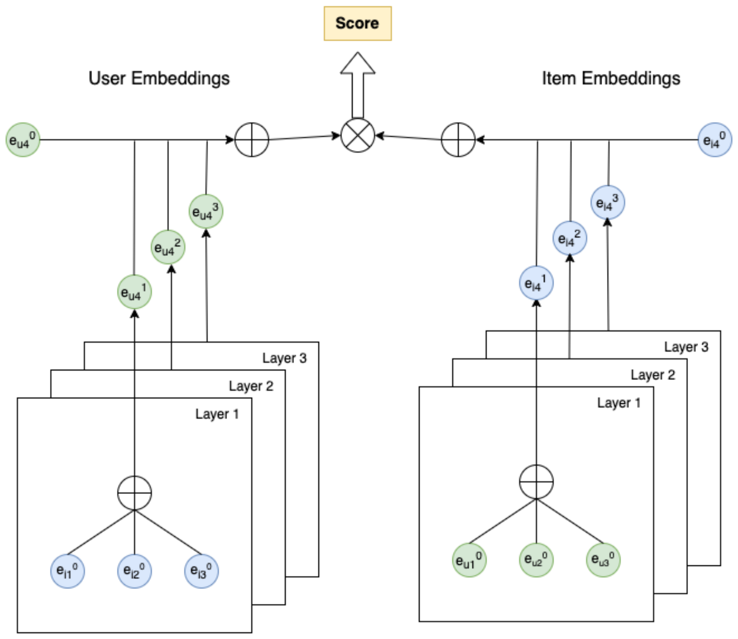

FIGURE 1. LightGCN Architecture

2.3. **DLRM.** DLRM is a deep neural network for content-based recommendations (Figure 2). The inputs to the model are the features of user u and item i, and the output is a prediction for $r(u, i)$.

In the first layer, the continuous features are fed through a multi-layer perceptron, with d total output neurons. In addition, each of the c categorical features is fed through an embedding layer of dimension d.

The second layer is a feature interaction layer that does not have any learnable parameters. At the end of the first layer, we have $c + 1$ total embeddings, each of dimension d. The feature interaction layer will calculate dot-products between pairs of embeddings, creating a $(c+1) \times (c+1)$ matrix. The upper triangle of this matrix, including the diagonal, will be flattened and returned as a single vector of length $\frac{(c+1)(c+2)}{2}$.

The final layer is another multi-layer perceptron that maps the vector produced by the feature interaction layer to a single neuron that regresses the predicted ranking for the user and item.

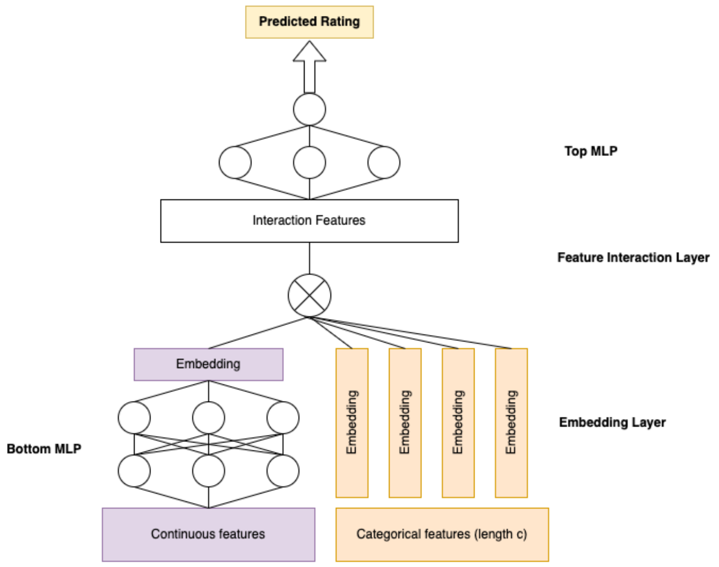

FIGURE 2. DLRM Architecture

2.4. **Data Pre-Processing.**

2.4.1. *LightGCN.* First, I read the MovieLens-1M ratings into a Pandas dataframe. Then, I filtered the dataframe to only include ratings of value 5, since I chose to define a ”positive” user-item relationship as the user having rated the item a 5. Then, I performed an 80-20 train-test split on the data. Next, I created a list of undirected edges based on the ratings data to use as the graph shift operator.

The pre-processing for the Amazon Fashion dataset was almost exactly the same. However, I experimented with different levels of sparsity in the graph. So, in my preprocessing, I added an initial step to only include users with at least m ratings, where m was a hyperparameter.

2.4.2. *DLRM.* First, I read the ratings data, the movie features, and user features into a single merged dataframe. Then, I encoded the categorical features to ordinal values. I then performed an 80-20 train/test split using the same random seed as for LightGCN. I extracted TF-IDF features from the titles of the movies, adding the top 20 most relevant features to the dataset. After this, I had a dataset with $c = 5$ categorical variables (user age range, user gender, user occupation, user zipcode, movie genre) and 20 continuous features (the 20 TF-IDF features for movie title).

2.5. **Training.**

2.5.1. *LightGCN.* At each training step, I randomly sampled a batch of users. For each user, I would also select one item they had a positive interaction with (i.e. an item they have an edge to) and an item they have not had a positive interaction with (i.e. they do not have an edge to). I trained the network using the Bayesian Personalized Ranking (BPR) Loss: $l(w) = \frac{1}{b}\sum_{i=1}^{b} \sigma(u_i \cdot v_i^p - u_i \cdot v_i^n)$, where b is batch size, σ is the softplus activation function (to keep the loss positive), u_i is the output embedding for user i, and v_i^p and v_i^n are the output embedding for user i's positive and negative item. I also added weight decay to the initial embedding layer.

2.5.2. *DLRM.* At each training step, I randomly sampled a batch of user-product ratings from the training set. I trained the network with Mean Squared Error loss on the predicted rating. I also added weight decay on all parameters of the network.

2.6. **Evaluation.** I evaluated the models by generating 5 recommendations per user. My north-star metric was "hit percentage," which measures the proportion of users who were recommended at least one item that they truly rated 5/5. I also calculated recall and precision, the traditional metrics for recommendations, but favored hit percentage because recall and precision are heavily influenced by the number of recommendations given per user and the number of known ratings per user. For LightGCN, I generated the top-5 recommendations simply by taking the dot product of each user's embeddings with each item's embeddings and returning the top 5 highest dot product scores per user. For DLRM, I generated the top-5 recommendations by predicting the rating that each user would give to each item and returning the top 5 highest predicted items per user.

3. Experimental Results

3.1. **LightGCN on MovieLens-1M.** I trained LightGCN on MovieLens-1M for 8 epochs, using a batch size of 32, an embedding dimension of 32, 3 graph convolutional layers, and a weight decay parameter of 0.001 (Appendix Figure 6). These combination of parameters achieved the best performance, with a test hit-percentage of 51.6% (Figure 3). The model had recall of 13.1% and precision of 28.9%, an improvement over existing work. This performance is solid, especially since we only give 5 total recommendations per user. Interestingly, the model was showing signs of overfitting, since the hit percentage on the training set was over 90%. However, experimenting with higher weight decay constants worsened performance on both the training and test set – for example, weight decay of 0.01 led to a 47% test hit percentage, and only an 82% train hit percentage.

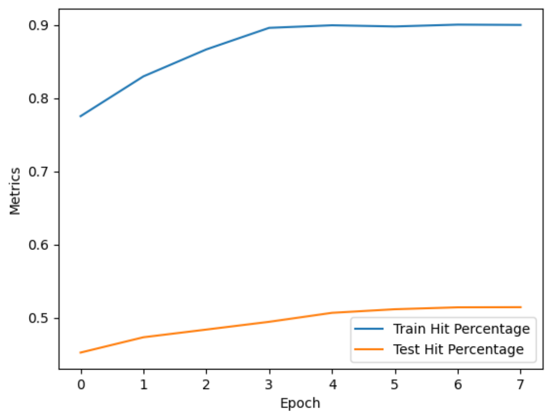

FIGURE 3. Hit Percentage of LightGCN on MovieLens-1M

3.2. **Varying LightGCN Embedding Dimension.** I investigated how different embedding dimensions of 8, 16, 32, 64, and 128 would affect LightGCN's performance. Neither increasing nor decreasing the embedding dimension significantly affected performance. Dimension 8 had the lowest hit percentage on the test set, but it was still 48.7%, reasonably close to the optimum, achieved with embedding dimension 32. I believe that there is a minimum embedding dimension necessary to sufficiently create clusters of users and items, but the model hits diminishing marginal returns with large embeddings due to the curse of dimensionality and instabilities during training.

3.3. **LightGCN on Amazon Fashion Reviews.** I evaluated the performance of LightGCN on Amazon Fashion Reviews, varying the number of ratings per user in the training set to see how LightGCN's performance changed. The results clearly showed that a larger number of reviews per user in the training set led to better performance, with all metrics increasing as sparsity decreased (Figure 4). This indicates that collaborative filtering does not work well when little information is known about users, which is in-line with prior literature. Even at 6 ratings per user, Amazon Fashion is a much sparser dataset than MovieLens, which has minimum 20 ratings per user, and we see that its hit percentage is lower than for MovieLens.

3.4. **DLRM on MovieLens-1M.** The best performing DLRM model had 18M total parameters, with 15 layers in both the top and bottom MLPs and an embedding dimension of 512. I trained the model for 30 epochs with a batch size of 256 and weight decay of 0.001. The final model had a train hit percentage of 46.5% and a test hit percentage of only 20.8% (Figure 5). I tried varying the number of layers and the hidden dimension and noticed that the model needed at least 10 layers in each MLP for the train hit percentage to even get above 20%.

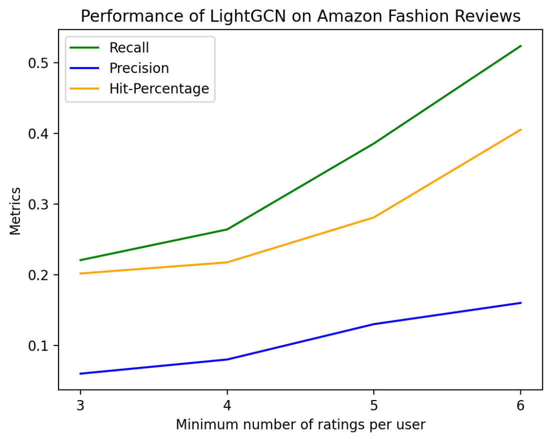

FIGURE 4. Performance of LightGCN on Amazon Fashion

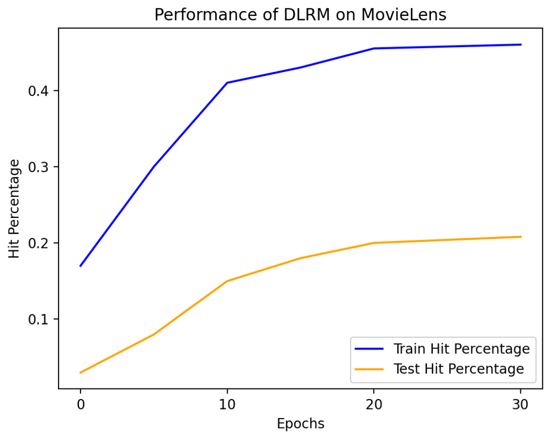

FIGURE 5. Hit Percentage of DLRM on MovieLens

4. DISCUSSION

My results provide valuable context into the strengths and weaknesses of collaborative filtering and content-based approaches for recommendation systems.

LightGCN demonstrated powerful capabilities on both MovieLens and Amazon Fashion, pointing to the utility of simple graph convolutional networks for collaborative filtering. Notably, LightGCN did not require a large embedding dimension to accurately learn user-item preferences. However, its performance degraded heavily as the dataset became more sparse. This indicates that collaborative filtering falters when making recommendations to users or items with little history. In addition, LightGCN's architecture requires a learnable embedding for each node in the user-item graph, meaning it may not be feasible in a setting where there are millions or billions of users and items, since this would lead to an intractably large model.

DLRM did not perform as well as LightGCN on MovieLens. It required a large number of layers and parameters to reach half of the hit percentage of LightGCN. In addition, it has very high latency when making recommendations for users, since we have to do many compute-intensive forward passes through the network to predict ratings. However, because it is a content-based approach, it does not suffer from the sparsity issues that collaborative filtering does. This could make it more applicable in settings where there are lots of users and items, and few ratings per user. Furthermore, it is important to note that the results in this work are isolated to MovieLens: it could be that the user and movie features do not contain much signal.

If I had more time, I would like to modify LightGCN to include weighted edges for all ratings, not just 5/5, since this would capture much more information from the dataset. I would also like to benchmark DLRM on a different dataset that contains more rich user and item features.

5. References

1. Dong, Z., Wang, Z., Xu, J., Tang, R., amp; Wen, J. (2022, September 5). A brief history of Recommender Systems. arXiv. https://arxiv.org/abs/2209.01860

2. Gupta, U., et al. The Architectural Implications of Facebook's DNN-based Personalized Recommendation. arXiv. https://arxiv.org/pdf/1906.03109.pdf

3. Li, Y., Liu, K., Satapathy, R., Wang, S., amp; Cambria, E. (2023, June 22). Recent developments in Recommender Systems: A survey. arXiv.org. https://arxiv.org/abs/2306.12680

4. Gao, C., Zheng, Y., Li, N., Li, Y., Qin, Y., Piao, J., Quan, Y., Chang, J., Jin, D., He, X., amp; Li, Y. (2023, January 12). A survey of graph neural networks for recommender systems: Challenges, methods, and directions. arXiv.org. https://arxiv.org/abs/2109.12843

5. He, X., Deng, K., Wang, X., Li, Y., Zhang, Y., amp; Wang, M. (2020, July 7). LIGHTGCN: Simplifying and powering graph convolution network for recommendation. arXiv.org. https://arxiv.org/abs/2002.02126

6. Huang, W, et al. (2023) Dual-LightGCN: Dual light graph convolutional network for discriminative recommendation. Computer Communications. https://www.sciencedirect.com/science/article/pii/S014036642300097X

7. Naumov, M., Mudigere, D., Shi, H.-J. M., Huang, J., Sundaraman, N., Park, J., Wang, X., Gupta, U., Wu, C.-J., Azzolini, A. G., Dzhulgakov, D., Mallevich, A., Cherniavskii, I., Lu, Y., Krishnamoorthi, R., Yu, A., Kondratenko, V., Pereira, S., Chen, X., … Smelyanskiy, M. (2019, May 31). Deep learning recommendation model for personalization and recommendation systems. arXiv.org. https://arxiv.org/abs/1906.00091

8. F. Maxwell Harper and Joseph A. Konstan. 2015. The MovieLens Datasets: History and Context. ACM Transactions on Interactive Intelligent Systems (TiiS) 5, 4, Article 19 (December 2015), 19 pages. DOI=http://dx.doi.org/10.1145/2827872

9. Ni, J., et al. (2019). Justifying recommendations using distantly-labeled reviews and fined-grained aspects. Empirical Methods in Natural Language Processing. https://nijianmo.github.io/amazon/index.html

6. Appendix

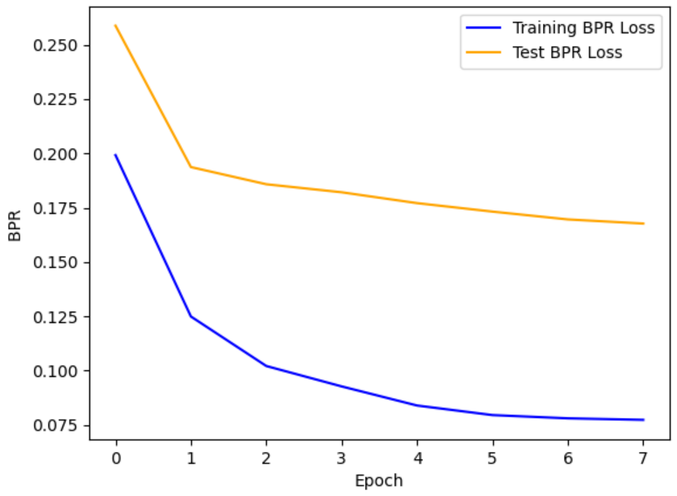

FIGURE 6. BPR Loss of LightGCN on MovieLens