Article Title

Frederik Valdemar Schrøder, Jens Petur Tróndarson, Mathias Møller Lybech

Department of Computer Science, Aalborg University

fschra16@student.aau.dk jtrand16@student.aau.dk mlybec16@student.aau.dk

March 2, 2021

Abstract

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

1. Introduction

Lorem ipsum dolor sit amet, consectetur adipiscing elit.

1.1. Pre specialization paper

In preparation for our master thesis, we spent a semester gaining knowledge in the field of recommendation systems. We wrote a paper called *Exploring Recommender Systems with Graph Convolutional Networks*. In this paper, we studied recommendation systems using GCN's to achieve state of the art results. In particular, we looked at the models Neural Graph Collaborative Filtering (NGCF), Price-aware User Preference-modeling (PUP) and LightGCN. PUP and LightGCN are both inspired by NGCF, where LightGCN removes complexity operations from NGCF and PUP adds side information in form of prices and categories. Both achieved better performance in NDCG and recall compared to NGCF. We were however not able to achieve the results, where PUP performed better than NGCF in our own experiments. Because of this, we thought it would be interesting to extend LightGCN with price and categories to see if it would improve performance.
Our initial extension of LightGCN where we changed the input by adding prices and categories to the user-item graph only made the model perform worse. It is our hypothesis that further changing how the embedding is performed can lead to performance increases.

2. Method

2.1. Brief Review of NGCF, LightGCN and GCF

Neural Graph Collaborative Filtering (NGCF) was published in 2019, and was a state of the art method, that utilized graph convolutional networks (GCN) for collaborative filtering [5]. GCN was originally proposed for node classification where each node has rich attributes, but for collaborative filtering the nodes only contain node IDs [2, 3, 5]. This is something that LightGCN took advantage of and simplified NGCF by removing the nonlinear activation function and the feature transformation, which showed promising results in their experiments. Later Meng Liu et. al then created a method called BiTGCF which utilized GCN and transfer learning between two domains [4]. They also added GCF which is a degenerate version of BiTGCF that does not utilize transfer learning between two domains, but is a single domain method such as LightGCN and NGCF. BiTGCF and GCF removes the features transformation and nonlinear activation function of NGCF, which was inspired by Light-

GCN, but chooses to keep different aspect of NGCF. In their experiments Meng Liu et. al showed that both GCF and BiTGCF showed large improvements over both LightGCN and NGCF, but BiTGCF only showed smaller improvements over GCF, which can indicate that high connectivity caused by GCN has larger effect on recommendation than transfer learning between other domains. In this paper we only concentrate on single domain methods, and therefore will only focus on GCF of these two methods.

2.1.1 Graph Convolutional Networks

GCN for collaborative filtering can generally be abstracted as [4]

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

where $AGG(\cdot)$ is an aggregation function, and $e_i^{(k)}$ and $e_u^{(k)}$ denotes the embeddings for user u and item i after k propagation layers. The embedding propagation for items can be obtained similarly as the embeddings with users by replacing u with i and vice versa. $\mathcal{N}_u$ and $\mathcal{N}_i$ denotes the one hop neighbors from user u and item i respectively [2, 4, 5].
On Figure 1 a generalized figure of LightGCN, GCF and NGCF can be seen. Graph convolutions are conducted on the initial embeddings, where an embedding function is used. The graph convolutions are illustrated on Figure 2 and this figure shows how the signal from $e_{i_1}^{(0)}$ passes through $e_{u_3}^{(1)}$ and $e_{i_4}^{(2)}$ to $e_{u_1}^{(3)}$. This is how the graph convolutions are performed on all users and items. When the convolutions have been conducted on each layer, it is given as input to the layer combination, which could either be concatenation or weighted summation. An example can be seen on Figure 3, where each array symbolizes a layer. Following this section we will describe the embedding propagation for users and the final layer combination for NGCF, LightGCN and GCF.

2.1.2 Embedding Propagation in NGCF:

The embedding propagation in NGCF is defined as,

$$\mathbf{e}_u^{(k+1)} = \text{LeakyReLU}(\mathbf{m}_{u\leftarrow u}^{(k)} + \sum_{i\in\mathcal{N}_u} \mathbf{m}_{u\leftarrow i}^{(k)}), \tag{2}$$

where LeakyReLU is the nonlinear activation function, $\mathbf{m}_{u\leftarrow i}^{(k)}$ is the message construction from users u to item i, and $\mathbf{m}_{u\leftarrow u}^{(k)}$ is the self connection. These are respectively defined in Equation 3 and Equation 4 [5].

$$\mathbf{m}_{u\leftarrow i}^{(k)} = \frac{1}{\sqrt{|\mathcal{N}_u||\mathcal{N}_i|}}(\mathbf{W}_1^{(k)}\mathbf{e}_i^{(k)} + \mathbf{W}_2^{(k)}(\mathbf{e}_i^{(k)} \odot \mathbf{e}_u^{(k)})), \tag{3}$$

$$\mathbf{m}_{u\leftarrow u}^{(k)} = \mathbf{W}_1^{(k)}\mathbf{e}_u^{(k)} \tag{4}$$

For Equation 3 and Equation 4 W_1 and W_2 are the trainable weight matrices used to perform feature transformation at each layer. $\frac{1}{\sqrt{|\mathcal{N}_u||\mathcal{N}_i|}}$ is the graph Laplacian norm used to normalize the embeddings. The final embedding is calculated by concatenating each embedding layer as seen on Equation 5 [5].

$$\mathbf{e}_u = \mathbf{e}_u^{(0)}||...||\mathbf{e}_u^{(k)} \tag{5}$$

2.1.3 Embedding Propagation in LightGCN

The embedding propagation for LightGCN is defined as [2],

$$\mathbf{e}_u^{(k+1)} = \sum_{i\in\mathcal{N}_u} \frac{1}{\sqrt{|\mathcal{N}_u||\mathcal{N}_i|}}\mathbf{e}_i^{(k)} \tag{6}$$

As can be seen, LightGCN simplified NGCF by removing the activation function, transformation matrices, the self loop and the inner product between the user and item embeddings. Additionally LightGCN changes the layer combination from concatenation to weighted summation as shown here:

$$\mathbf{e}_u = \sum_{k=0}^{K} \alpha_k \mathbf{e}_u^{(k)}, \tag{7}$$

where K is the total number of layers and α_k is a hyper parameter used to denote the importance of the k-th embedding [2]. α_k is set to $1/(K+1)$ for simplicity.

2.1.4 Embedding Propagation in GCF.

The embedding propagation in GCF is defined as [4],

$$\mathbf{e}_u^{(k+1)} = \mathbf{e}_u^{(k)} + \sum_{i\in\mathcal{N}_u} \frac{1}{\sqrt{|\mathcal{N}_u||\mathcal{N}_i|}}\left(\mathbf{e}_i^{(k)} + \mathbf{e}_i^{(k)} \odot \mathbf{e}_u^{(k)}\right) \tag{8}$$

As can be seen GCF adds the self connections and the inner product between the users and items compared to LightGCN. The purpose of the inner product is that the more the user and item nodes have in common the greater the value will be passed to the center node [4]. Self connection is used to retain the information of the original node. Additionally GCF uses concatenation as layer combination, which was also used by NGCF in equation Equation 5.

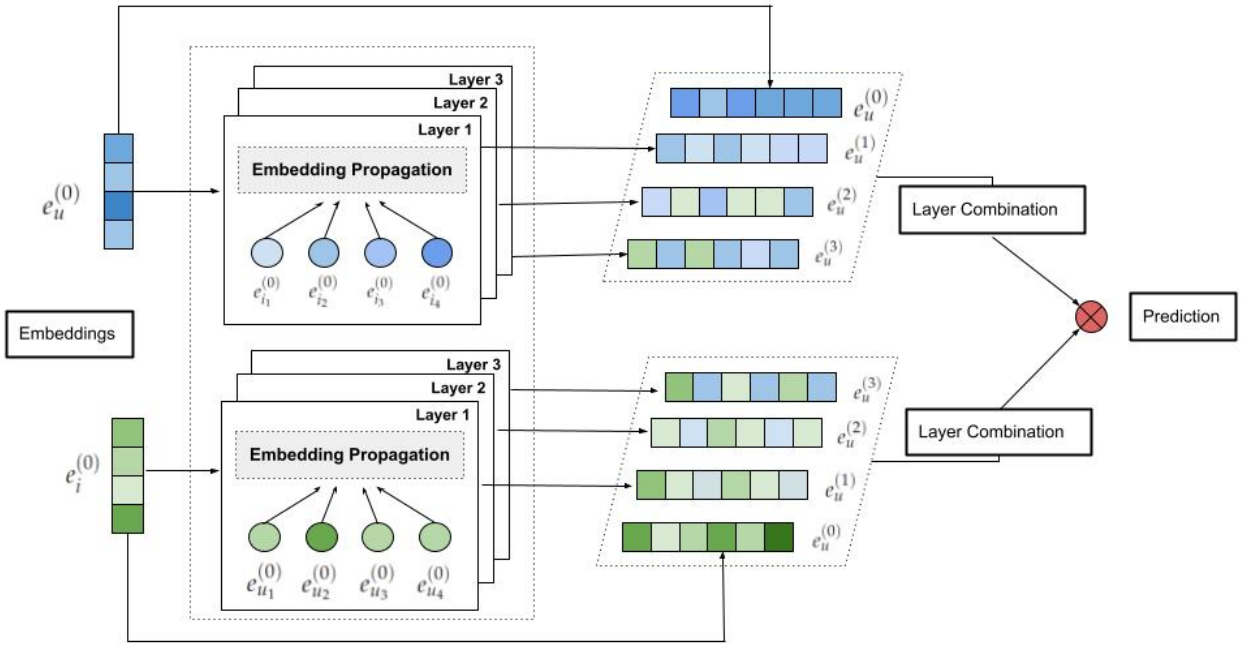

Figure 1: *Generalized structure of NGCF, LightGCN and GCF*

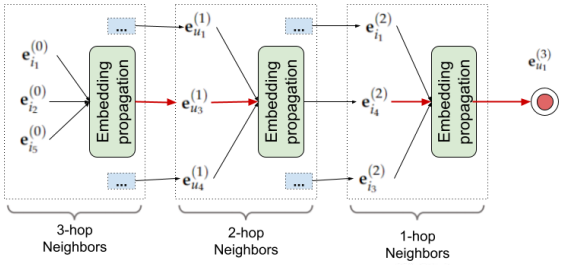

Figure 2: *Illustration of three graph convolutions for* u_1

Layer Combination
LightGCN: Weighted sum([3,4,6], [2,6,8], [1,5,7]) = [2, 5, 7]
NGCF / GCF: Concat([3,4,6], [2,6,8], [1,5,7]) = [3,4,6,2,6,8,1,5,7]

Figure 3: *Example of weighted summation and concatenation as layer combinations.*

2.2. Optimizing layer combination

2.2.1 Removing α_k

Before trying to optimize α_k we wanted to investigate what impact it had. The simplest experiment is to change the value of α_k to 1 which essentially removes it. If this results in small changes in performance, it could indicate that it would not be worthwhile to pursue the opportunity to optimize α_k.

$$\mathbf{e}_u = \sum_{k=0}^{K} \mathbf{e}_u^{(k)}, \tag{9}$$

The result of the experiment can be seen in Section 3.2.1

2.2.2 Optimizing α_k

The initial idea of optimizing α_k was to change each layer's effect depending on how many connections each node had. An example can be seen Table 1. The numbers shown on the table are arbitrary and different variables need to be tested to see how well it performs. The intuition behind this is that the effect of each layer in the GCN being dependent on the number of connections that the node has would be beneficial. A node with only 1 connection will

Number of connections	Effect of each layer
0 - 20	10 % effect of $e^{(0)}$ 15 % effect of $e^{(1)}$ 25 % effect of $e^{(2)}$ 50 % effect of $e^{(3)}$
21-50	10 % effect of $e^{(0)}$ 10 % effect of $e^{(1)}$ 60 % effect of $e^{(2)}$ 20 % effect of $e^{(3)}$
50+	10 % effect of $e^{(0)}$ 50 % effect of $e^{(1)}$ 20 % effect of $e^{(2)}$ 20 % effect of $e^{(3)}$

Table 1: *Example of the changed effect to α_k.*

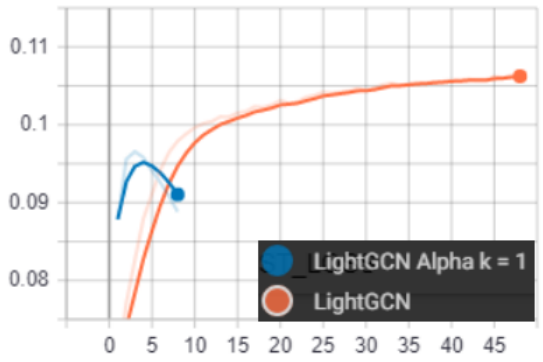

Figure 4: *NDCG@50 of LightGCN and LightGCN-Ak1 on yelp2020*

not benefit much from the first convolution but will be richer in information in the third convolution. For a node with 200 connections, the third convolution might actually be harmful, as the node gets impacted too much by too many nodes, where 1 convolution is enough to make it perform optimally.

3. Experiment

3.1. Datasets

For our experiments we have used 3 different datasets. These are *yelp-2020*, *amazon-book* and *amazon-cell-sport*. The data split is 20% testing and 80% training where 10% of the training data is used as validation data. A comparison of the datasets can be seen on Table 2.

The *amazon-book* dataset is the largest and simplest dataset that we used. It is taken from the [2] paper and only contains ID's for users and items. As in [2] we have used the 10-core setting *i.e.* only removing users and items with less than ten interactions.

The *amazon-cell-sport* dataset is taken from [4]. This dataset contains two domains *cell* short for Cell Phones and *sport* short for Sports and Outdoors. It contains two domains because [4] builds an extension of [2] which utilizes a cross-domain recommendation technique. To build a cross-domain dataset first all the overlapping users are found. Then all the users with more five or more interactions are retained. Items from each domain have both been split into training and testing before being merged.

The *yelp-2020* dataset was made as a compromise to test both [2] and [1]. PUP needs both price and category in addition to user and item IDs so a dataset was needed which could be used to validate both methods. Both methods used a different version of Yelp's datasets which were incompatible so utilizing Yelp's newest dataset (at the time) was decided. Inspired by PUP the dataset only retains businesses belonging to the restaurant category where each restaurant have one or more sub-categories such as Mexican, Chinese, etc. To make the data simpler only one sub-category is retained for each item, this is what is considered its category in the rest of the text.

3.2. Changing α_k

3.2.1 Removing α_k

In this experiment we changed α_k from $(1/(K+1))$ to 1 and essentially removed the variable to see what impact it had on the results. This method is called LightGCN-Ak1 which stands for LightGCN $\alpha_k = 1$. This will result in the embeddings scaling more than they otherwise would have with α_k as normalization. As can be seen on Figure 4 and Figure 5 changing α_k decreases performance of LightGCN on the yelp2020 dataset. In the initial epochs, LightGCN-Ak1 performs better, but quickly starts to decline in performance. On Figure 6 and Figure 7 LightGCN and LightGCN-Ak1 was run on the amazon-book dataset, but can seen that LightGCN still outperforms LightGCN-Ak1. These results indicates that α_k is an important part of the layer combination for weighted summation, and we will continue our experimentation on trying to optimize α_k

	Users	Items	Items/users ratio	Interactions	Sparsity
Amazon-cell	4,998	14,618	2.924	47,444	99.9351%
Amazon-sport	4,998	22,101	4.421	55,556	99.9497%
Amazon-cell-sport	4,998	36,719	7.347	103,000	99.9438%
Amazon-book	52,643	91,599	1.74	2,984,108	99.9381%
Yelp2020	24,384	20,091	0.824	594,196	99.8787%

Table 2: *Difference between the datasets*

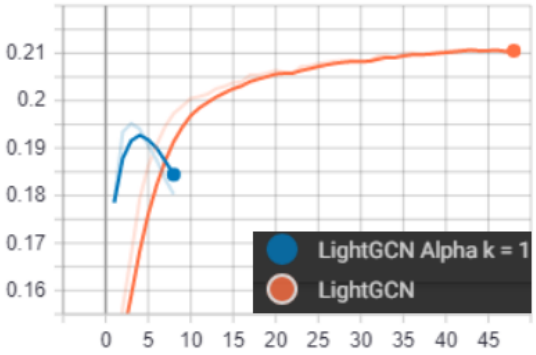

Figure 5: *Recall@50 of LightGCN and LightGCN-Ak1 on yelp2020*

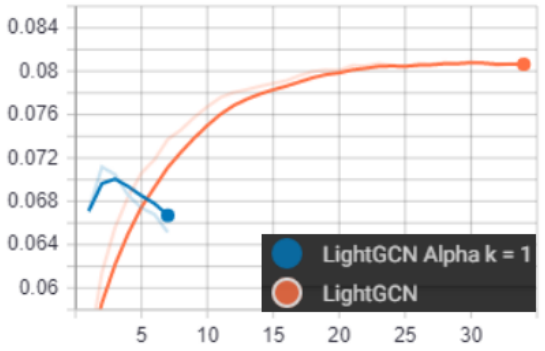

Figure 7: *Recall@50 of LightGCN and LightGCN-Ak1 on amazon-book*

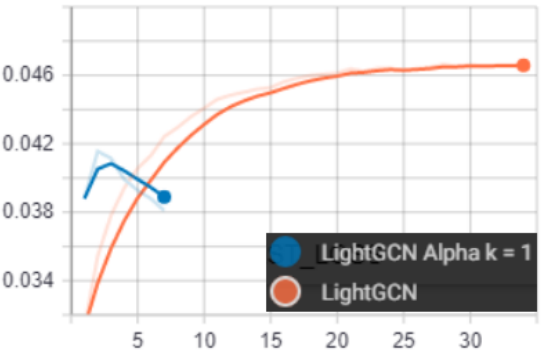

Figure 6: *NDCG@50 of LightGCN and LightGCN-Ak1 on amazon-book*

Appendices

Here should the first input to the appendix be

References

[1] Yu Zheng et al. "Price-aware Recommendation with Graph Convolutional Networks". In: *KDD '18* (2020).

[2] Xiangnan He et al. "LightGCN: Simplifying and Powering Graph Convolution Network for Recommendation". In: SIGIR '20. 2020.

[3] Thomas N. Kipf and Max Welling. "Semi-Supervised Classification with Graph Convolutional Networks". In: (2017). arXiv: `1609.02907 [cs.LG]`.

[4] Meng Liu et al. “Cross Domain Recommendation via Bi-Directional Transfer Graph Collaborative Filtering Networks”. In: (2020).

[5] Xiang Wang et al. “Neural Graph Collaborative Filtering”. In: *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval* (2019).