Performance Aware Layer Combination for Graph Convolutional Networks

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

June 1, 2021

Abstract

Graph Convolutional Networks (GCN) is a State-of-the-Art method used for recommendation. Throughout this paper we suggest two new alternative extensions for LightGCN called Aggressive Layer Combination (ALC) and Balanced Layer Combination (BLC). These are used as alternative layer combination methods for graph convolutional networks instead of using LightGCNs version of weighted summation. This showed large improvements on datasets where items have few connections and users have a larger amount of connections. An ablation study for GCF is also conducted to understand why it outperformed LightGCN on their datasets. ALC and BLC both outperformed GCF. Additionally, we experimented with some edge cases, where only one embedding layer was used or where the 0th embedding is removed. Only utilizing one embedding layer also showed in many cases to outperform ALC, BLC, GCF and LightGCN.

1. Introduction

Recommendation systems aim to alleviate the problem of information overload when browsing the web. They are widely used for e.g. online shopping where the number of items available can be overwhelming. By looking at the data of the current user the recommendation system can predict a certain number of items that it thinks the user would like, thereby greatly reducing the unnecessary information shown. There are different ways to achieve this where one of the most common ways is using collaborative filtering.

Collaborative filtering is based on the concept that users that act similarly will most likely have the same preferences, meaning users who have bought the same things and liked the same things will in the future most likely have an interest in the same products. Achieving collaborative filtering is done by learning latent features of items and users to represent them, this is also called embeddings. One of the earlier models for collaborative filtering is Matrix Factorization (MF) where the users and items are embedded as vectors and the dot-product of the vectors is then used to make predictions of which items users would like [9].

More recent collaborative filtering models utilize graph convolutional networks (GCN's) to better capture the collaborative signals. Models like Neural Graph Collaborative Filtering (NGCF) integrate the bipartite graph structure, that is the user-item-interactions, into the embedding process to capture the collaborative signals[16]. A later model called LightGCN simplifies NGCF, removing feature transformation and the activation function. Doing this achieved state-of-the-art performance, as they showed that feature transformation and activation functions were not beneficial for recommendation [3].

One of the major components of GCN's is the layer combination. GCN's start with a user-item graph and performs convolutions on the graph to create additional embeddings for each layer. In the end, these layers are combined to create the final embedding, the most common ways to combine these layers is by using concatenation or the weighted

summation.
To our knowledge, there has not been done any work on optimizing this process. In this paper we have looked into what effect changing the layer combination method has on the performance of the LightGCN model. We have come up with different methods called Aggressive Layer Combination (ALC) and Balanced Layer Combination (BLC) which rank how much effect each layer should have in the final embedding to achieve better performance. We have also seen that utilizing only one layer and disregarding the rest can also lead to performance increase.
In this paper, we have also done an ablation study to better understand how GCF the non-transfer learning version BiTGCF can outperform LightGCN in their paper.
These investigations can be summed up into the following research questions.

- **RQ1**: How does changing α_k affect the performance in LightGCN?
- **RQ2**: How do different aggregation functions effect the performance in GCF and LightGCN?
- **RQ3**: How does adding ALC and BLC to LightGCN perform compared to other state of the art methods?
- **RQ4**: Is it beneficial to change layer combination based on the degree of the nodes?

2. Preliminary

This section contains the basic information needed to understand our contributions to LightGCN and knowledge about similar recommendation models. It explains how embedding propagation and layer combination is done in NGCF, LightGCN and GCF.

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 a GCN for collaborative filtering [16]. GCN was originally proposed for node classification where each node has rich attributes, but for collaborative filtering the nodes only contain node IDs [3, 7, 16]. 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 [12]. 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 like LightGCN and NGCF. BiTGCF and GCF remove the feature transformation and nonlinear activation function of NGCF, which was inspired by LightGCN, but they choose to keep different aspects of NGCF. In their experiments Meng Liu et. al showed that both GCF and BiTGCF had 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 a larger effect on recommendation than transfer learning between domains has. 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 [12]

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

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 convolutions. 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 [3, 12, 16].
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.

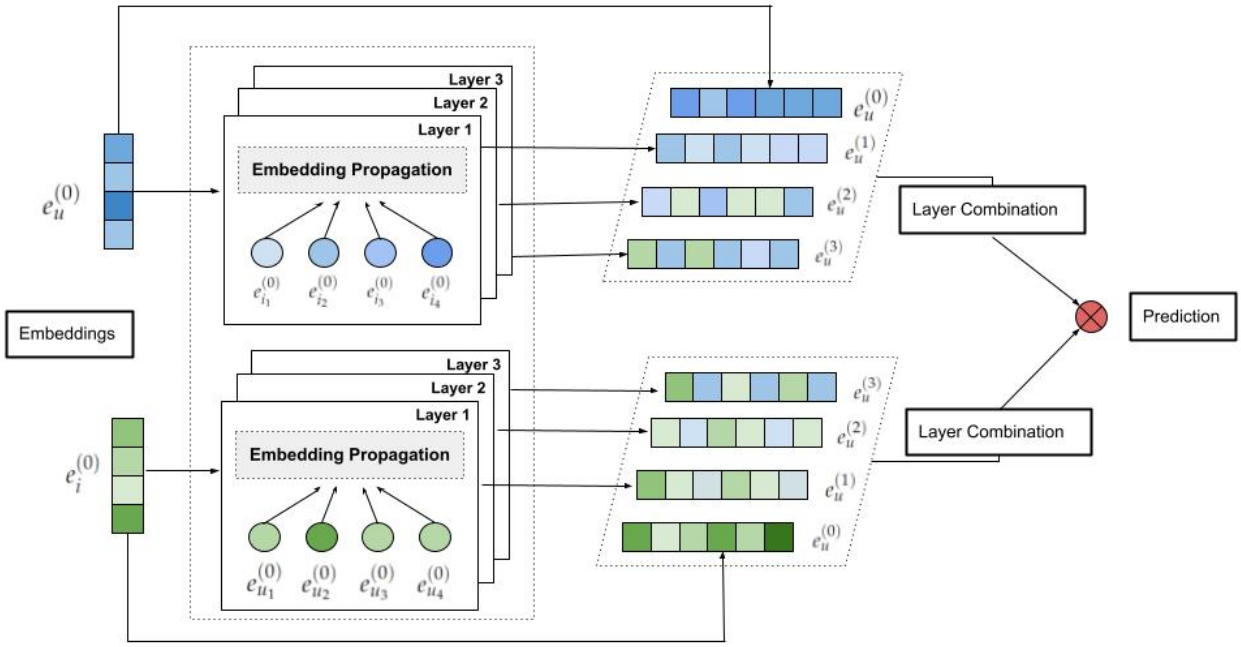

Figure 1: *Generalized structure of NGCF, LightGCN and GCF*

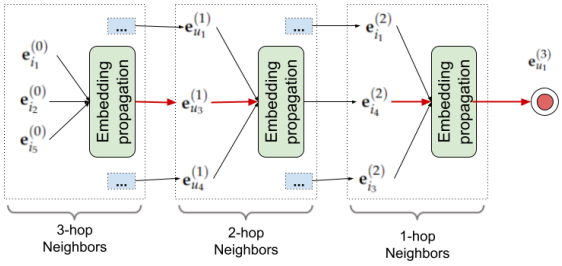

Figure 2: *Illustration of three graph convolutions for u_1*

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.*

2.1.2 Embedding Propagation in NGCF:

The embedding propagation in NGCF is defined as,

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

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 [16].

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

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

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 [16].

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

2.1.3 Embedding Propagation in LightGCN

The embedding propagation for LightGCN is defined as [3],

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

As can be seen, LightGCN simplified NGCF by removing the activation function, learnable weight 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 [3]. 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 [12],

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

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 [12]. 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.

2.2. Removing α_k

The first method that we tried was removing α_k. If this did not have any substantial effect on the performance of LightGCN, it could indicate that it would not be worthwhile trying to optimize the layer combination method. This was done by changing α_k to 1, which equates to summation of the different layer embeddings.

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

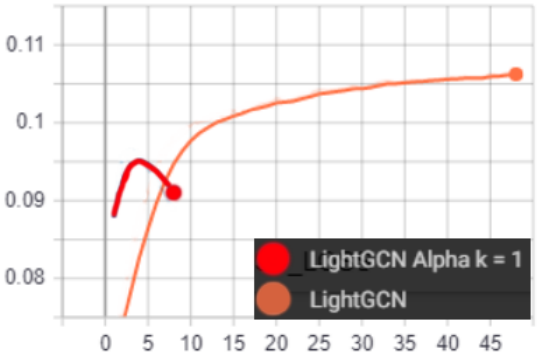

Figure 4: *NDCG@50 of LightGCN and LightGCN-Ak1 on yelp2020*

This will make the embeddings scale when they are combined instead of normalizing them as they do in LightGCN. The results from running the experiment on the yelp2020 dataset can be seen on Figure 4. It shows that removing α_k was detrimental for the performance. In the initial epochs, LightGCN $\alpha_k = 1$ (LightGCN-Ak1) performs better, but it quickly starts to decline in performance. The results from amazon-book are similar and can be seen in Section A.1 These results indicates that α_k is an important part of the layer combination for weighted summation, and that summation is not beneficial for LightGCN.

3. Method

Throughout this section we introduce ALC and BLC as our contribution to LightGCN. These methods experiment with different changes to how layer combination is performed in LightGCN. We also describe some edge cases when changing α_k such as removing the 0th embedding and only utilizing one layer.

3.1. Optimizing layer combination

In this subsection we describe the methods that are mentioned in the following research question:

- **RQ3**: How does adding ALC and BLC to LightGCN perform compared to other state of the art methods?

LightGCN uses weighted summation as their aggregation function when combining the different layers, where each layer has the same weight. The hypothesis is that some layers are more important than

others therefore giving each layer the same effect can decrease the performance of the model. Instead, we have developed two algorithms that take into account the performance of each layer to see if there is a substantial performance gain to be had when optimizing the layer combination. Given a dataset, a GCN model will have a number of convolution layers. The algorithms looks at the performance of each layer compared to the best performing layer. Based on how much worse the layer performs compared to the best layer its effect on the final embedding will be reduced and the other layers will be increased.

3.1.1 Aggressive Layer Combination

The pseudo-code for the aggressive layer combination (ALC) algorithm can be seen on algorithm 1. Each layer starts out with the same effect. Given the NDCG score of each layer, the algorithm looks at how many percent worse each layer performed compared to the best layer. The layers are then sorted with the worst performing layer first. For each layer, the amount of percent it has performed worse is subtracted from its effect and shared among the remaining layers to increase their effect. After this, the layer is popped from the list and can not regain any effect. This can leave multiple layers with having zero effect on the final embedding.
This might not be optimal as even layers who might not perform well on their own can still provide some kind of collaborative signal. This is seen in experiments where the 0th embedding was removed and yielded a worse result even though its individual result was poor. Therefore a more balanced algorithm was thought to be better.

```
Result: A list of how much effect each layer
        has on the final embedding
L = Ordered list of performance of each layer
effect = List of current effect for each layer
while i < L.length; i++ do
    effect [i] = 100 / L.length
end
while L.length > 1; i++ do
    worseP = L.pop()
    newEffect = max(Inf[i] - worseP, 0)
    while k < L.length; k++ do
        effect[k] = effect[k] + ((effect[i] -
          newEffect ) / L.length)
    end
    effect[i] = newEffect
end
return effect.orderBy(layerId)
```

Algorithm 1: Algorithm for the aggressive layer combination based on performance

3.1.2 Balanced Layer Combination

The pseudo-code for the balanced layer combination (BLC) algorithm can be seen on algorithm 2. The BLC algorithm works much the same as ALC but instead of popping a given layer of the list when its effect has been reduced, it is instead kept. This means that when reducing other layers effect some of their effects will be given to the already reduced layer. This insures no layer will ever reach zero effect as can happen in ALC.

Examples of how the aggressive and balanced algorithms work can be seen on Figure 5 and Figure 6 In both examples, we have 4 layers from 0 to 3 and each layer has an initial effect of 25%. Their performance is color-coded to make it easier to follow which values are used to calculate the new effect.

```
Result: A list of how much effect each layer
        has on the final embedding
L = Ordered list of performance of each layer
effect = List of current effect for each layer
while i < L.length; i++ do
    effect [i] = 100 / L.length
end
while i = 0; i < L.length; i++ do
    worseP = L.[i]
    newEffect = max(effect[i] - worseP, 0)
    while k = 0; k < L.length; k++ do
        if(k != i) effect[k] = effect[k] +
          ((effect[i] - newEffect ) / (L.length-1))
    end
    effect[i] = newEffect
end
return effect.orderBy(layerId)
```

Algorithm 2: Algorithm for the balanced layer combination based on performance

Layers	e(0)	e(1)	e(2)	e(3)
Worse performance %	0	3	9	15
Start effect %	25	25	25	25
Iteration 1	25+(15/3)	25+(15/3)	25+(15/3)	25-15
Current effect %	30	30	30	10
Iteration 2	30+(9/2)	30+(9/2)	30-9	
Current effect %	34.5	34.5	21	
Iteration 3	34.5+(3/1)	34.5-3		
Current effect %	37.5	31.5		
Final effect %	37.5	31.5	21	10

Figure 5: *Example of aggressive splitting of layer effect based on performance*

Layers	e(0)	e(1)	e(2)	e(3)
Worse performance %	0	3	9	15
Start effect %	25	25	25	25
Iteration 1	25+(15/3)	25+(15/3)	25+(15/3)	25-15
Current effect %	30	30	30	10
Iteration 2	30+(9/3)	30+(9/3)	30-9	10+(9/3)
Current effect %	33	33	21	13
Iteration 3	33+(3/3)	33-3	21+(3/3)	13+(3/3)
Current effect %	34	30	22	14
Final effect %	34	30	22	14

Figure 6: *Example of balanced splitting of layer effect based on performance*

The yellow squares mark which layer is getting a lower effect in the current iteration and green marks the specific layer's final effect. Comparing these two algorithms it can be seen that there is a larger diversion in the aggressive algorithm than in the balanced algorithm.

3.2. The effect of α_k in LightGCN

This sections serves to answer the following research question:

- **RQ1**: How does changing α_k affect the performance in LightGCN?

We experimented with three different methods to see how changes to how layer combination is performed could effect LightGCN's performance. This includes removing α_k, only using 1 layer and removing the 0th layer embedding. Only utilizing 1 layer is required to calculate ALC and BLC to know the performance of each layer. Removing the 0th embedding is a likely occurrence for ALC.

3.2.1 Utilizing only one layer

The second method that we tried was only utilizing one layer. This was done by performing the graph convolutions and then only using the embeddings from one of the layers instead of combining them. This can be seen in Equation 10 where k is the layer that we choose to utilize. $\mathbf{e}^{(0)}$ will have zero convolutions, $\mathbf{e}^{(1)}$ will have one convolutions, $\mathbf{e}^{(2)}$ will have two convolutions and so on.

$$\mathbf{e}_u = \mathbf{e}_u^{(k)} \tag{10}$$

3.2.2 Removing 0th layer

The third method that we tried was removing 0th layer. In this method we used the normal layer combination method used in LightGCN, but we did not include the 0th layer embedding in the final embedding as seen in Equation 11.

$$\mathbf{e}_u = \sum_{k=1}^{K} \alpha_k \mathbf{e}_u^{(k)}, \tag{11}$$

where K is the number of layers and α_k is set to $1/K$.

3.3. Model training

LightGCN utilizes Bayesian Personalized Ranking (BPR) loss which is a pairwise loss that takes implicit feedback (e.g. clicks or purchased items) and ranks observed interactions in the prediction higher than the unobserved counterparts [3, 14]. BPR assumes that observed interactions are more reflective on a users preference than the unobserved counterparts, and hereby assigns observed items a higher value. This loss is used to update the only learnable parameter, which is $\mathbf{E}^{(0)}$.

$$L_{BPR} = -\sum_{u=1}^{M} \sum_{i \in \mathcal{N}_u} \sum_{j \notin \mathcal{N}_u} \ln \sigma(\hat{y}_{ui} - \hat{y}_{uj}) + \lambda ||\mathbf{E}^{(0)}||^2 \tag{12}$$

In Equation 12 λ is L_2 regularization which is used to prevent overfitting. σ is the sigmoid function. Adams optimizer is utilized and is used with mini-batches [3, 5]. Adams optimizer uses momentum and adaptive learning rates in contrast to stochastic gradient descent [5]. Adams requires a batch of randomly sampled triples of users, observed item and unobserved item (u, i, j) and utilizes the calculated loss from this to update $E^{(0)}$. These model parameters are updated by the gradient of the loss function [16, 5]. $E^{(k)}$ is also updated, but is simply derived from $E^{(k-1)}$.

4. Evaluation

Throughout this section we perform experiments to answer the following research questions:

- **RQ1**: How does changing α_k affect the performance in LightGCN?
- **RQ2**: How do different aggregation functions effect the performance in GCF and LightGCN?
- **RQ3**: How does adding ALC and BLC to LightGCN perform compared to other state of the art methods?
- **RQ4**: Is it beneficial to change layer combination based on the degree of the nodes?

4.1. Datasets

For our experiments, we have used 6 different datasets. These are *Yelp2020*, *Amazon-Book*, *Amazon-Cell-Sport*, *Amazon-Cell-Electronic*, *Amazon-Cloth-Electronic* and *Amazon-Cloth-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 1.
Amazon-Book is the largest dataset we have used. It is taken from the LightGCN paper and only contains ID's for users and items [3]. LightGCN have used the 15-core setting for users and 5 core settings for items *i.e.* removing users with less than 15 interactions and items with less than 5 interactions.
The *Yelp2020* dataset is a dataset we used in our pre-specialization paper to mimic a dataset used in LightGCN that we could not get access to. This dataset uses 10-core settings for users and 5 core settings for items. It is a subset of the full yelp2020 dataset where all entries are restaurants with one category and price attribute.
Amazon-Cell-Sport*, *Amazon-Cell-Electronic*, *Amazon-Cloth-Electronic* and *Amazon-Cloth-Sport are taken from BiTGCF [12]. These datasets contain two domains because BiTGCF builds an extension of LightGCN which utilizes a cross-domain transfer learning recommendation model [12]. To build a cross-domain dataset first all the overlapping users were found. Then all the users with five or more interactions were retained. Items from each domain have both been split into training and testing before being merged. *Amazon-Cell-Sport*, *Amazon-Cloth-Electronic* and *Amazon-Cloth-Sport* use 5 core settings for users and 1 core settings for items. There is an exception for *Amazon-Cell-Electronic* which uses 15 core settings for users.

4.1.1 Interactions in datasets

As we expect there is a relation between the number of connections for each node and the performance of a convolution layer, we analyzed the different datasets used for our experiments. On Table 2 and Table 3 the amount of users and items that have a certain amount of interactions can be seen. Users in *Amazon-Book* have an average node degree of 45.22 and items have an average node degree of 25.99 connections, which makes it the dataset with the highest average node degree compared to the other datasets. For Yelp2020 the average node degree is 19.04 for users and 23.31 for items. The other four amazon datasets have in common that the average node degree for items is small. Users in *Amazon-Cell-Sport* have an average node degree of 17.07 and items have an average node degree of 2.58. *Amazon-Cell-Electronic* has an average node degree of 42.22 for users and an average node degree 2.73 for items. *Amazon-Cloth-Electronic* has an average node degree of 19.04 for users and an avevrage node degree of 3.12 for items. *Amazon-Cloth-Sport* has an average node degree of 16.72 for users and an average node degree of 2.49 for items.

4.2. Experimental Settings

4.2.1 Baselines

To see the effectiveness of our methods we compare them to the following baselines:

- **NGCF** [16]: was created for collaborative filtering utilizing a Graph Convolutional Network. Further description can be seen in Section 2.1.
- **LightGCN** [3]: was created from NGCF by removing weights, activation function and changing the layer combination to weighted summation. Further description can be seen in Section 2.1.
- **GCF** [12]: is a degenerate method of BiTGCF that does not utilize transfer learning. GCF showed to outperform LightGCN in their experiments.
- **GCN** [8]: was created with the purpose of semi-supervised node classification with Graph Convolutional Networks.
- **GC-MC** [1]: is a graph-based auto-encoder framework for matrix completion that produces latent features for users and items with message passing.

	Users	Items	Items/users ratio	Interactions	Sparsity
Amazon-Book	52,643	91,599	1.74	2,984,108	99.9381%
Yelp2020	24,384	20,091	0.824	594,196	99.8787%
Amazon-Cell-Sport	4,998	36,719	7.347	103,000	99.9438%
Amazon-Cell-Electronic	3,325	57,925	17.42	172,611	99.9103%
Amazon-Cloth-Electronic	15,761	105,174	6.673	363,474	99,9780%
Amazon-Cloth-Sport	9,928	73,613	7.414	200,297	99,9726%

Table 1: *Comparisons on the datasets*

	Yelp2020				Amazon-Cell-Sport				Amazon-Book			
Range	Users		Items		Users		Items		Users		Items	
1-5	0	0 %	5185	26.02 %	0	0 %	30152	91.31 %	0	0 %	4393	4.79 %
6-10	7298	29.92 %	4652	23.34 %	1190	23.80 %	1677	5.07 %	0	0 %	20562	22.44 %
11-15	7175	29.42 %	2505	12.57 %	1890	37.81 %	565	1.71 %	0	0 %	21561	23.53 %
16-20	3862	15.83 %	1622	8.14 %	936	18.72 %	270	0.817 %	17098	32.47 %	12092	13.20 %
21-25	1742	7.14 %	1135	5.69 %	396	7.92 %	137	0.41 %	8234	15.64 %	7638	8.33 %
26-30	1160	4.75 %	814	4.08 %	211	4.22 %	78	0.23 %	5257	9.98 %	5092	5.55 %
31-35	749	3.07 %	624	3.13 %	115	2.30 %	44	0.133 %	3722	7.07 %	3791	4.13 %
36-40	664	2.72 %	487	2.44 %	78	1.56 %	27	0.081 %	3154	5.99 %	2891	3.15 %
41-45	380	1.55 %	398	1.99 %	46	0.92 %	19	0.057 %	2029	3.85 %	2301	2.51 %
46-50	243	0.99 %	323	1.62 %	31	0.62 %	14	0.042 %	1591	3.02 %	1775	1.93 %
51-60	409	1.67 %	475	2.38 %	36	0.72 %	13	0.039 %	2581	4.90 %	2407	2.62 %
61-70	200	0.82 %	316	1.58 %	19	0.38 %	5	0.015 %	1664	3.16 %	1662	1.81 %
71-80	126	0.51 %	271	1.36 %	13	0.26 %	6	0.018 %	1379	2.61 %	1183	1.29 %
81-90	102	0.41 %	199	0.99 %	5	0.10 %	4	0.012 %	954	1.81 %	883	0.96 %
91-100	71	0.29 %	145	0.72 %	6	0.12 %	0	0 %	745	1.41 %	619	0.67 %
101-150	127	0.52 %	434	2.17 %	23	0.46 %	6	0.018 %	2120	4.02 %	1528	1.66 %
151-200	52	0.21 %	179	0.89 %	2	0.04 %	1	0.003 %	892	1.69 %	553	0.60 %
201-250	17	0.06 %	64	0.32 %	0	0 %	0	0 %	469	0.89 %	251	0.27 %
251-300	1	0.004 %	36	0.18 %	0	0 %	0	0 %	254	0.48 %	146	0.15 %
300+	7	0.028 %	61	0.30 %	1	0.02 %	0	0 %	500	0.94 %	271	0.29 %
Avg con	19.04		23.31		17.07		2.58		45.22		25.99	

Table 2: *Amount of nodes within a certain node degree for Amazon-Cell-Sport, Amazon-Book and Yelp2020. The Avg connection shows how many connections each user or item have on average*

	Amazon-Cell-Electronic				Amazon-Cloth-Electronic				Amazon-Cloth-Sport			
Range	Users		Items		Users		Items		Users		Items	
1-5	0	0.0	46183	89.94	0	0.0	85169	88.81	0	0.0	61217	91.85
6-10	0	0.0	3072	5.982	3345	21.22	6505	6.783	2331	23.479	3603	5.406
11-15	0	0.0	990	1.928	5625	35.68	1944	2.027	3675	37.016	979	1.468
16-20	538	16.180	477	0.928	2957	18.76	862	0.898	1853	18.664	364	0.546
21-25	685	20.601	204	0.397	1358	8.616	429	0.447	916	9.226	168	0.252
26-30	477	14.345	128	0.249	774	4.910	243	0.253	443	4.462	110	0.165
31-35	334	10.045	63	0.122	458	2.905	170	0.177	244	2.457	56	0.084
36-40	250	7.518	61	0.118	295	1.871	131	0.136	149	1.500	53	0.079
41-45	177	5.323	42	0.081	208	1.319	97	0.101	95	0.956	32	0.048
46-50	146	4.390	29	0.056	157	0.996	63	0.065	64	0.644	17	0.025
51-60	194	5.834	40	0.077	186	1.180	67	0.069	62	0.624	23	0.034
61-70	144	4.330	16	0.031	107	0.678	51	0.053	37	0.372	6	0.009
71-80	98	2.947	12	0.023	83	0.526	41	0.042	16	0.161	4	0.006
81-90	60	1.804	13	0.025	48	0.304	29	0.030	17	0.171	5	0.007
91-100	44	1.323	7	0.013	37	0.234	15	0.015	11	0.110	2	0.003
101-150	100	3.007	8	0.015	68	0.431	45	0.046	10	0.100	5	0.007
151-200	41	1.233	2	0.003	37	0.234	14	0.014	4	0.040	0	0.0
201-250	16	0.481	0	0.0	12	0.076	11	0.011	0	0.0	0	0.0
251-300	10	0.300	0	0.0	2	0.012	2	0.002	1	0.0100	0	0.0
300+	11	0.330	1	0.0019	4	0.025	7	0.007	0	0.0	0	0.0
Avg con	42.22		2.73		19.04		3.12		16.72		2.49	

Table 3: *Amount of nodes within a certain node degree for Amazon-Cell-Electronic, Amazon-Cloth-Electronic and Amazon-Cloth-Sport. The Avg connection shows how many connections each user or item have in average*

4.2.2 Parameter settings

BiTGCF and GCF utilized the binary cross entropy loss function as this works well for cross domains, but as we only investigate GCF, we decided to change this to BPR to be able to compare this with LightGCN and NGCF. For all experiments the embedding size is 64 and mini batch size is 2048 on Yelp2020 and Amazon-Cell-Sport. For Amazon-Book the mini batch size is set to 8192 for speed. LightGCN is optimized with Adam [6] and the learning rate is set to 0.001 and dropout is deactivated. The embedding parameters are initialized with Xavier method [2, 3]. Early stopping is used and the maximum number of epochs is set to 1000.

4.3. GCF ablation study

This subsection focus' on answering the following research question:

- **RQ2:** How do different aggregation functions effect the performance in GCF and LightGCN?

Meng Liu et. al does not present an ablation study for BiTGCF and GCF, which LightGCN showed is important when studying the different components in NGCF [3, 12]. GCF outperformed LightGCN on all datasets used in BiTGCF [12], and we would like understand which parts of GCF causes the increase in performance. The GCF aggregation function is changed by either changing the layer combination to weighted summation, removing the inner product, removing self connections or only utilizing the inner product. Examples of the changed methods can be seen in the following equations. *GCF-minus-sc* can be seen on Equation 13 where the self-connection has been removed. On Equation 14 only the inner product of the neighbour has been preserved, and is called *GCF-only-IP*. In Equation 15 the inner product has been removed and is called *GCF-minus-IP*. There are also examples where GCF utilizes summation as layer combination as used in LightGCN which can be seen on Equation 7. The methods that use weighted summation all start with *GCF-sum*. When LightGCN is tested with concatenation as layer combination it is called *LightGCN-concat*.

$$\mathbf{e}_u^{(k+1)} = \sum_{i \in \mathcal{N}_u} \frac{1}{\sqrt{|\mathcal{N}_u||\mathcal{N}_i|}} \left(\mathbf{e}_i^{(k)} + \mathbf{e}_i^{(k)} \odot \mathbf{e}_u^{(k)} \right) \quad (13)$$

$$\mathbf{e}_u^{(k+1)} = \mathbf{e}_u^{(k)} + \sum_{i \in \mathcal{N}_u} \frac{1}{\sqrt{|\mathcal{N}_u||\mathcal{N}_i|}} \mathbf{e}_i^{(k)} \odot \mathbf{e}_u^{(k)} \quad (14)$$

$$\mathbf{e}_u^{(k+1)} = \mathbf{e}_u^{(k)} + \sum_{i \in \mathcal{N}_u} \frac{1}{\sqrt{|\mathcal{N}_u||\mathcal{N}_i|}} \mathbf{e}_i^{(k)} \quad (15)$$

We did not include equations for all of the methods to reduce redundancy, but the method names and descriptions are as follows:

- **GCF**: The original GCF method as described in Section 2.1.4.
- **GCF-minus-sc**: GCF without self connections.
- **GCF-only-IP**: GCF where $e_i^{(k)}$ has been removed in Equation 8, so that GCF's graph convolutions only considers the inner product of users and items.
- **GCF-only-IP-minus-sc**: Implemented as GCF-only-ip but without self connections.
- **GCF-minus-IP**: GCF where inner product has been removed.
- **LightGCN-concat**: LightGCN with concatenation as layer combination.
- **LightGCN**: Original LightGCN as described in Section 2.1.3.
- **LightGCN-plus-sc**: LightGCN, but with self connections.
- **GCF-sum-only-IP**: Implemented as GCF-only-IP except that the layer combination method used is weighted summation.
- **GCF-sum**: GCF where the layer combination has been changed to weighted summation instead of concatenation.
- **GCF-sum-minus-sc**: Implemented as GCF-sum but without self connections.

The results can be seen on Table 4, where the bold results are the best performing and underlined are the second best results. Graphs of how the performance of the methods change over epochs can be seen in Figure 7, Figure 8 and fig:GCF-NDCG-ablation-study-amazon-book. These are for the datasets yelp2020, amazon-book and amazon-cell-sport respectively. More comprehensive figures can be seen in B, where the summation methods and concatenation methods are in separate figures.

4.3.1 Concatenation and weighted summation

Looking at Yelp2020 and Amazon-Book on Table 4 the methods that utilize concatenation as their layer combination method generally perform worse than the methods that utilize summation. On Figure 7 most concatenation methods learn faster than the summation methods, but the concatenation methods are also prone to early stopping because they start to decline in performance. This is also the case

	Yelp2020		Amazon-Cell-Sport		Amazon-Book	
	NDCG@50	Recall@50	NDCG@50	Recall@50	NDCG@50	Recall@50
GCF	0.09092	0.1869	0.03398	0.06536	0.04032	0.07035
GCF-minus-sc	0.09084	0.1879	**0.03472**	**0.06656**	0.04108	0.07261
GCF-minus-ip	0.09179	0.1881	0.03197	0.06294	0.03977	0.06998
GCF-only-ip	0.07659	0.1587	0.01818	0.03832	0.03765	0.06607
GCF-only-ip-minus-sc	0.08338	0.1712	0.02578	0.05535	0.03777	0.06621
LightGCN-concat	0.0856	0.1735	0.03029	0.05707	0.03798	0.06519
LightGCN	**0.1064**	**0.2106**	0.033	0.06278	0.04675	0.08129
LightGCN-plus-sc	0.1031	0.2098	0.03212	0.06261	**0.04679**	**0.08175**
GCF-sum	0.09724	0.1988	0.03095	0.06446	0.04075	0.07205
GCF-sum-minus-sc	0.0956	0.1962	0.03075	0.0629	0.04114	0.07261
GCF-sum-only-ip	0.09843	0.199	0.02878	0.06065	0.04114	0.07212

Table 4: *NDCG and Recall of the changed methods.*

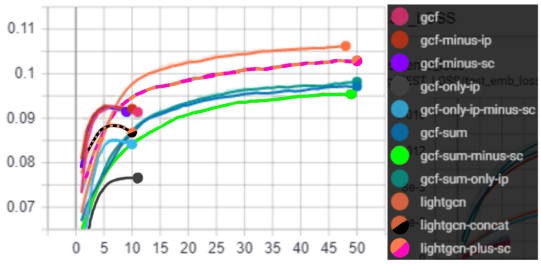

Figure 7: *NDCG@50 for the Yelp2020 dataset.*

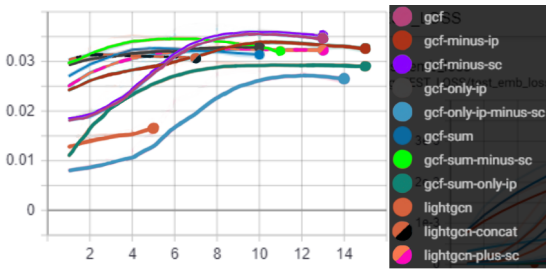

Figure 8: *NDCG@50 for the Amazon-Sport-Cell dataset.*

for Figure 9, although some concatenation methods are not prone to early stopping. On Figure 8 the methods perform differently compared to the Yelp2020 dataset. The summation methods train for a lower number of epochs compared to the Yelp2020 dataset. This could be because amazon-cell-sport is a smaller dataset than Yelp2020. Generally it can be seen that GCF and GCF-minus-sc perform better than the other methods in Figure 8. LightGCN performs better than the GCF methods that utilize weighted summation as their layer combination method. However with Amazon-Cell-Sport the results vary less between the methods, although *gcf-sum-only-ip* is clearly performing worse than the other methods.

4.3.2 Inner product

Methods which use weighted summation perform worse over time when utilizing inner product. The only exception is Recall@50 on Amazon-Cell-Sport for *GCF-sum* which performs better than LightGCN. This could also simply be because the inner prod-

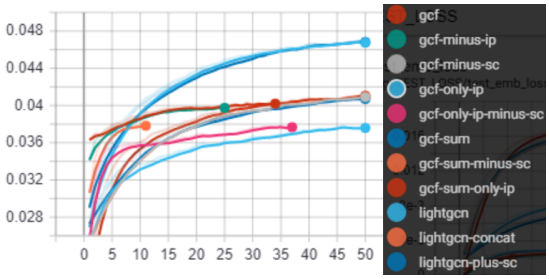

Figure 9: *NDCG@50 for the Amazon-Book dataset.*

uct in general is beneficial for datasets where users or items have few connections. For *GCF* and *GCF-minus-ip* it makes a small difference to add inner product in Yelp2020 and Amazon-Book, however for Amazon-Cell-Sport the method using inner product performs 3.8 % better for Recall@50 and 6 % better for NDCG@50. These result can indicate that utilizing inner product is favorable for datasets with few connections.

4.3.3 Self connections

When comparing the counter parting methods on Table 4 that either utilize or do not utilize self connections there is often a minimal difference on the results. For LightGCN, GCF and GCF sum, using self connections makes a small difference. For *LightGCN* and *LightGCN-plus-sc* it varies which one performs best, but *GCF-minus-sc* outperforms *GCF* by a small amount most of the time. This is probably dependent on how many connections there are in the dataset. It seems datasets with few connections declines in performance by adding self connections, and datasets with many connections benefit from adding self connections. This would probably be because when the node has few connections, the self connection will have a large influence on the embedding. Interestingly *gcf-only-ip-minus-sc* performs significantly better than *gcf-only-ip* on Yelp2020 and Amazon-Cell-Sport, which can indicate that self connections are harmful for performance, if you only use inner product in the convolutions. However on Amazon-Book it does not make a large difference.

4.3.4 Conclusion

The best performing methods are either GCF or LightGCN with or without self connections. GCF is the best performing on Amazon-Cell-Sport, which is the smallest datasets, that also includes a lot of items with very few interactions. We assume that a combination of inner product and concatenation is beneficial for learning on small datasets or datasets with few interactions between users and items. But on the larger datasets, LightGCN and LightGCN-plus-sc are the best performing methods.

4.4. Changing α_k

In this section we discuss the results from the experiments made with the methods that where described in Section 3.2. We answer the following research question in this subsection:

- **RQ1**: How does changing α_k affect the performance in LightGCN?

4.4.1 Utilizing only one layer

In this section we experiment with removing the layer combination, and only utilizing the embedding from a specific convolution layer. This method is described in Section 3.2.1. $\mathbf{e}^{(0)}$, $\mathbf{e}^{(1)}$, $\mathbf{e}^{(2)}$, $\mathbf{e}^{(3)}$, $\mathbf{e}^{(4)}$ and $\mathbf{e}^{(5)}$ are used as the final embeddings in each experiment. These results are used to calculate ALC and BLC. These are compared to LightGCN where weighted sum is utilized with a different number of layers. As can be seen on Table 5 the results vary a lot depending on the dataset. For Amazon-Cell-Sport only considering $\mathbf{e}^{(4)}$ and $\mathbf{e}^{(5)}$ gives the best results which perform around 13 % better than weighted sum with 5 convolutions. This could be because Amazon-Cell-Sport consists primarily of nodes with few interactions, and therefore the later convolutions have the largest impact. 90 % of all items within this dataset have 5 interactions, which is one of the reasons that the later convolutions perform so well. For Yelp2020 the best results were weighted sum with 5 convolutions closely followed by $\mathbf{e}^{(2)}$. This dataset varies more in terms of the number of interactions that the users have. For Amazon-Book weighted sum with 3 convolutions performs best and this could be because there is a large variation of how many interactions the users have. One of the advantages of weighted sum is that it stabilizes the results. This might have a positive effect in some cases compared to our method where we only use the embedding from one of the layers. From this we can conclude that for datasets, where items or users generally have few connections it is worth considering only using $\mathbf{e}^{(4)}$ or $\mathbf{e}^{(5)}$ instead of using weighted sum.

4.4.2 Removing 0th layer

The case that we described in Section 3.2.2 was removing the 0th layer embedding before combining the embeddings from the different layers. In this subsection we show results from the experiments done with this method. While Amazon-Cell-Sport has an improved performance when the 0th embedding is removed, the other datasets see a decrease in performance.

	Amazon-Cell-Sport		Yelp2020		Amazon-Book	
	NDCG@50	Recall@50	NDCG@50	Recall@50	NDCG@50	Recall@50
Weighted sum (1 con)	0.02804	0.05503	0.0969	0.1955	0.0427	0.07408
Weighted sum (2 con)	0.03132	0.06133	0.1008	0.2015	0.0463	0.08055
Weighted sum (3 con)	0.03237	0.06447	0.1064	0.2106	**0.04668**	**0.08129**
Weighted sum (4 con)	0.03253	0.06394	0.1084	0.2157	0.04617	0.08033
Weighted sum (5 con)	0.03285	0.06451	**0.1089**	**0.2177**	0.04515	0.07861
$\mathbf{e}^{(0)}$	0.02169	0.04447	0.08177	0.1674	0.03669	0.06373
$\mathbf{e}^{(1)}$	0.02523	0.04859	0.1019	0.2039	0.0458	0.079
$\mathbf{e}^{(2)}$	0.03419	0.06809	0.1086	0.217	0.04487	0.07755
$\mathbf{e}^{(3)}$	0.03483	0.06972	0.09956	0.2001	0.0372	0.06412
$\mathbf{e}^{(4)}$	0.0366	**0.07377**	0.08863	0.1788	0.03247	0.05607
$\mathbf{e}^{(5)}$	**0.03733**	0.07318	0.0819	0.1643	0.02923	0.05022

Table 5: *Experiment on LightGCN where different layers are used as the final embedding compared with weighted sum.*

	Recall@50		NDCG@50	
Method	5 con average	$E^{(0)}$ removed	5 con average	$E^{(0)}$ removed
Amazon-Cell-Sport	0.06451	**0.06726**	0.03285	**0.03460**
Yelp2020	**0.2177**	0.21289	**0.1089**	0.10641
Amazon-Book	**0.08129**	0.07715	**0.04668**	0.04470

Table 6: *Results from experiment where we remove 0th layer embedding*

4.5. Performance Comparison with State-of-the-Arts

In the following subsection experiments are conducted to answer the following research question:

- **RQ3**: How does adding ALC and BLC to LightGCN perform compared to other state of the art methods?

Table 7 and Table 8 shows the results of our experiments with NDCG@50 and Recall@50 respectively. We compare our method to LightGCN, NGCF, GC-MC and GCN. All experiments were done with 5 convolutions. Amazon-Book actually performed better with 3 convolutions, but because ALC and BLC were calculated using 5 convolutions we decided to compare it with amazon-book using 5 convolutions. ALC and BLC could possibly perform better on Amazon-Book with fewer convolutions. The primary observations from the experiments are:

- ALC, BLC and $\mathbf{e}^{(i)}$ generally perform better than all other methods, except for on Amazon-Cell-Electronic and Amazon-Cloth-Sport. Amazon-Cell-Electronic differentiates from other datasets because it has an item/user ratio of 17.42, while other datasets have an item/user ratio below 8. Amazon-Cloth-Sport is similar to Amazon-Cell-Sport in terms of item/user ratio, however, it has twice as many users and items.
- ALC and $\mathbf{e}^{(i)}$ often perform better than BLC in NDCG@50, but BLC often performs better than ALC in Recall@50. For NDCG@50 $\mathbf{e}^{(i)}$ seems to either be best performing or close to performing best in most of the cases, except for on Amazon-Cloth-Sport.
- LightGCN outperforms all other baseline methods, except for GCF in Amazon-Cell-Sport. We seem to be unable to reconstruct that GCF outperforms LightGCN consistently as seen in their paper BiTGCF [12]. This could be because BiTGCF uses cross-entropy as their loss function for all methods while we use BPR. Another reason could be because they utilize the evaluation protocol of "Leave-one-out" and LightGCN uses "All-Ranking" protocol [12, 3]. GCF consistently outperforms NGCF and NGCF outperforms GCN and GC-MC most of the time. Between GCN and GC-MC it differs on the best performing depending on the dataset.
- We have not found any layer combination method that consistently can perform better than any other method. There is still work to

be done to find a layer combination method that can combine the layers best suited for the datasets.

4.6. Degree dependent layer combination

In this experiment we investigated how the nodes within a certain range of node degree perform. The node degree is the number of connections the node has in the original interaction graph. Our initial hypothesis was that the fewer connections a node had, the more it would benefit from a higher number of convolutions. However the results obviously confirm that this is not the case. We answer the following research question in this section:

- **RQ4**: Is it beneficial to change layer combination based on the degree of the nodes?

4.6.1 Only utilizing one layer

To be able to test ALC and BLC on different node degree ranges, we first had to find out how the different node degree ranges performed when only using one layer. The NDCG results can be seen for Amazon-Cell-Sport on Table 11 and Figure 11, Yelp2020 on Table 9 and Figure 10 and Amazon-Book on Table 10 and Figure 12. The results have been evaluated within a certain node degree, and "Combined" is where all nodes have been evaluated together to get the overall performance. The rightmost column is the result of the best performing original LightGCN method. All of the results for LightGCN with different layers with NDCG and Recall can be found in Appendix D. The tables and figures containing the results of the recall performance for the different degree ranges can be found in Appendix C. The experiments with the Amazon-Cell-Sport dataset can be seen on Table 11. These results showcase that for nodes with less than 46 connections $E^{(5)}$ is usually the best performing embedding and for nodes with more than 46 connections $E^{(5)}$ performs worse than other embeddings with a lower number of convolutions. $E^{(5)}$ does however often perform second best for many node degree ranges. It is also worth noting that $E^{(3)}$, $E^{(4)}$ and $E^{(5)}$ consistently outperform LightGCN with 5 convolutions. This could be because utilizing $E^{(0)}$ and $E^{(1)}$ is harmful for performance, because 91 % of all items only have between 1-5 connections and therefore the embeddings need more convolution to capture the collaborative signals. Looking at the yelp2020 and amazon-book datasets seen on Table 9 and Table 10 the results do not seem to be dependent on the degree of the nodes, as almost all node degree ranges either perform best or second best in $E^{(1)}$ and $E^{(2)}$. In both of these datasets nodes have above 19 connections on average and therefore the amount of collaborative signals increase much faster for each node in the graph compared to amazon-cell-sport. For Amazon-Cell-Sport the recall performance increases for the nodes with a higher degree, however for Yelp2020 and Amazon-Book the performance decreases for higher node degrees. This could be because the nodes with many connections get too much noise in Yelp2020 and Amazon-Book, but for Amazon-Cell-Sport there are many items with few connected users, and hereby it limits how much information users with many connections gain. The best result for "Combined" in Amazon-Book is 3 convolution with average. $E^{(1)}$ and $E^{(2)}$ significantly outperform the other embedding layers. For Yelp2020 the 5 convolution average performs best, while the $E^{(1)}$, $E^{(2)}$ and $E^{(3)}$ embedding layer are the best individual performing layer. For Amazon-Cell-Sport $E^{(2)}$, $E^{(3)}$, $E^{(4)}$ and $E^{(5)}$ outperform the 5 convolution average.
From these results we see an opportunity to change the layer combination in different ways to investigate if this will improve the results. The last method is to use Section 3.1 method to optimize the hyper parameters. As no direct link between the number of connections a user has and its performance was found, it could be interesting to look further into the number of implicit connections a user has once the convolutions have been performed. The conclusion on the results are that the individual degree of a node has a smaller effect on the final result than we initially expected.

4.6.2 Degree dependent ALC and BLC

Another opportunity for layer combination is to use ALC or BLC based on the node degree. So instead of using the combined performance results from each layer, then use the results within a certain split. These splits can be seen in Appendix Section D.1.

Beskriv videre når vi har resultaterne.

5. Related work

In this section, we will look at some related work that is relevant to our paper. We will start by looking

NDCG@50	NGCF	LightGCN	GCN	GC-MC	GCF	ALC	BLC	$\mathbf{e}^{(i)}$
Yelp2020	0.08502	0.1089	0.07594	0.07947	0.09092	0.10953	**0.11015**	0.1086 (2)
Amazon-Book	0.03811	0.04515	0.03268	0.03364	0.04032	0.04574	0.04537	**0.0458** (1)
Amazon-Cell-Sport	0.02476	0.033	0.02087	0.01709	0.03398	0.0356	0.03516	**0.03733** (5)
Amazon-Cloth-Sport	0.05826	0.06749	0.00996	0.01074	**0.07556**	0.05945	0.06356	0.06392 (2)
Amazon-Cell-Electronic	0.03399	0.05413	0.02589	0.0180	**0.05424**	0.05094	0.05399	0.05422 (3)
Amazon-Cloth-Electronic	0.00912	0.01710	0.01384	0.00829	0.01272	0.01941	0.01792	**0.02074** (5)

Table 7: *Performance comparison on NDCG@50 with different state of the art methods.*

Recall@50	NGCF	LightGCN	GCN	GC-MC	GCF	ALC	BLC	$\mathbf{e}^{(i)}$
Yelp2020	0.17535	0.2177	0.15317	0.16341	0.1869	0.21809	**0.21917**	0.217 (2)
Amazon-Book	0.06714	0.07861	0.05578	0.05826	0.07035	0.07919	**0.08066**	0.079 (1)
Amazon-Cell-Sport	0.05312	0.06451	0.04119	0.03723	0.06536	0.07002	0.06928	**0.07377** (4)
Amazon-Cloth-Sport	0.08501	0.10482	0.02070	0.02817	0.10385	0.10240	**0.10567**	0.10541 (2)
Amazon-Cell-Electronic	0.05248	0.07738	0.03668	0.02633	0.07165	0.07355	0.07846	**0.07909** (3)
Amazon-Cloth-Electronic	0.01863	0.03225	0.02575	0.01690	0.02948	0.03760	0.03506	**0.04061** (5)

Table 8: *Performance comparison on Recall@50 with different state of the art methods.*

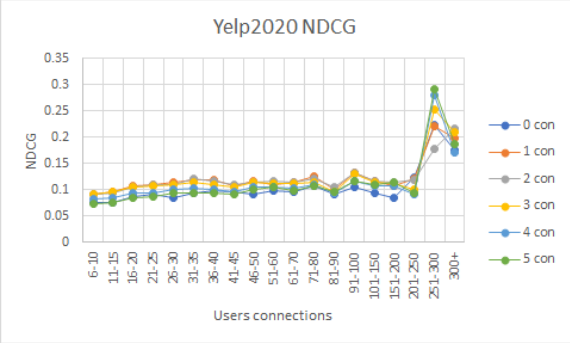

Figure 10: *NDCG performance for the individual embedding layers on Yelp2020*

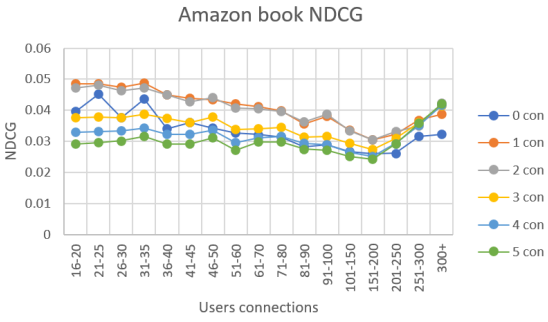

Figure 11: *NDCG performance for the individual embedding layers on Amazon-Book*

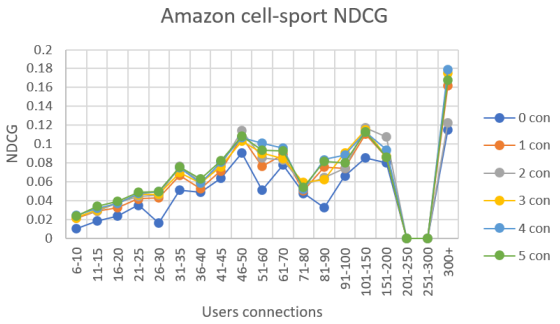

Figure 12: *NDCG performance for the individual embedding layers on Amazon-Cell-Sport*

at some work related to collaborative filtering and graph-based recommenders. Finally, we will look at some related work that also extended LightGCN or builds upon LightGCN in some capacity.

5.1. Collaborative filtering

Collaborative filtering(CF) is a widely used technique in modern recommender systems [3]. There are two main disciplines of CF which are the latent factor approach and the neighborhood approach [10]. Neighborhood approaches compute the relationship between items or alternatively between users. This will essentially transform users into sets of items that can then be compared. Latent factor

Node degree	$E^{(0)}$	$E^{(1)}$	$E^{(2)}$	$E^{(3)}$	$E^{(4)}$	$E^{(5)}$	5 con
6-10	0.07550	0.09271	0.09136	0.09211	0.08228	0.07408	**0.10095**
11-15	0.07677	0.09607	0.09444	0.09531	0.08449	0.07642	**0.10408**
16-20	0.08704	0.1069	0.10575	0.1043	0.09354	0.08562	**0.11451**
21-25	0.09096	0.1104	0.10892	0.1066	0.09455	0.08720	**0.11790**
26-30	0.08578	0.1137	0.10933	0.1092	0.1000	0.09431	**0.11909**
31-35	0.09420	0.1196	0.12151	0.1151	0.1021	0.09489	**0.12833**
36-40	0.09886	0.1195	0.11764	0.1108	0.09965	0.09326	**0.12711**
41-45	0.09512	0.1071	0.11056	0.1049	0.09713	0.09181	**0.11638**
46-50	0.09113	0.1162	0.11447	0.1147	0.1057	0.1011	**0.12537**
51-60	0.09782	0.1095	0.11560	0.1128	0.1059	0.1048	**0.12459**
61-70	0.09618	0.1133	0.11338	0.1125	0.1024	0.09921	**0.12377**
71-80	0.1067	0.1251	0.12072	0.1145	0.1096	0.1067	**0.13007**
81-90	0.09104	0.1019	0.10477	0.09683	0.09489	0.09547	**0.11042**
91-100	0.1045	0.1334	0.13189	0.1303	0.1177	0.1157	**0.14010**
101-150	0.09488	0.1168	0.11731	0.1142	0.1077	0.1092	**0.12676**
151-200	0.08451	0.1066	0.11441	0.1117	0.1065	0.1133	**0.11889**
201-250	**0.1231**	0.1217	0.11866	0.1009	0.09105	0.09490	**0.11999**
251-300	0.2238	0.2224	0.17787	0.2538	0.2806	**0.2917**	0.21693
301+	0.1752	0.1989	0.21585	0.2093	0.1719	0.1880	**0.23305**
Combined	0.08177	0.1019	0.1086	0.09956	0.08863	0.0819	**0.1089**

Table 9: *NDCG@50 for Yelp2020 where only one convolution layer is used and compared with the best performing LightGCN convolution for Yelp2020.*

Node degree	$E^{(0)}$	$E^{(1)}$	$E^{(2)}$	$E^{(3)}$	$E^{(4)}$	$E^{(5)}$	3 con
16-20	0.03965	0.04851	0.04722	0.03762	0.03287	0.02911	**0.04929**
21-25	0.03903	**0.04862**	0.04807	0.03788	0.03323	0.02964	0.04831
26-30	0.03767	**0.04751**	0.04624	0.03755	0.03335	0.03014	**0.04778**
31-35	0.03772	**0.04873**	0.04728	0.03865	0.03434	0.03163	0.04825
36-40	0.03412	**0.04506**	0.04495	0.03730	0.03227	0.02911	0.04493
41-45	0.03597	**0.04391**	0.04282	0.03607	0.03218	0.02916	0.04301
46-50	0.03436	0.04350	**0.04413**	0.03784	0.03357	0.03111	0.04368
51-60	0.03273	0.04205	0.04077	0.03380	0.02959	0.02719	**0.04210**
61-70	0.03214	**0.04109**	0.04054	0.03401	0.03117	0.02972	0.03972
71-80	0.03135	**0.03986**	0.03958	0.03445	0.03160	0.02981	0.03923
81-90	0.02832	0.03567	**0.03624**	0.03140	0.02928	0.02747	0.03551
91-100	0.02885	0.03814	**0.03882**	0.03163	0.02884	0.02717	0.03781
101-150	0.02675	**0.03351**	0.03332	0.02927	0.02652	0.02522	0.03334
151-200	0.02594	0.03037	**0.03049**	0.02728	0.02518	0.02423	0.02932
201-250	0.02613	0.03228	**0.03312**	0.03096	0.02925	0.02914	0.03242
251-300	0.03153	**0.03682**	0.03492	0.03594	0.03508	0.03581	0.03617
301+	0.03219	0.03874	**0.04227**	0.04137	0.04169	0.04207	0.04130
Combined	0.03669	0.0458	0.04487	0.0372	0.03247	0.02923	**0.04668**

Table 10: *NDCG@50 for Amazon-Book*

Node degree	$E^{(0)}$	$E^{(1)}$	$E^{(2)}$	$E^{(3)}$	$E^{(4)}$	$E^{(5)}$	5 con
6-10	0.00995	0.02156	0.02250	0.02173	**0.02441**	0.02355	0.01972
11-15	0.01814	0.02845	0.02901	0.02986	0.03154	**0.03376**	0.02984
16-20	0.02378	0.03269	0.03737	0.03739	0.03674	**0.03923**	0.03701
21-25	0.03508	0.04174	0.04424	0.04620	0.04727	**0.04907**	0.04438
26-30	0.03584	0.04281	0.04606	0.04663	0.04928	**0.04967**	0.04815
31-35	0.05091	0.06659	**0.07622**	0.06981	0.07514	0.07544	0.07514
36-40	0.04884	0.05221	0.06001	0.05867	0.05855	**0.06266**	0.05475
41-45	0.6337	0.07144	0.07698	0.07592	0.08052	**0.08197**	0.07634
46-50	0.09049	0.10755	**0.11442**	0.1030	0.1067	0.1083	0.10074
51-60	0.05128	0.07620	0.08498	0.09005	**0.1005**	0.09309	0.08827
61-70	0.07796	0.08878	0.08324	0.08425	**0.09537**	0.09270	0.07718
71-80	0.04767	0.05050	0.05611	**0.05930**	0.05158	0.05406	0.05834
81-90	0.03265	0.07549	0.06440	0.06200	**0.08332**	0.08163	0.07351
91-100	0.06619	0.07376	0.07435	**0.09016**	0.08820	0.08012	0.08968
101-150	0.08542	0.11063	**0.11732**	0.1151	0.1130	0.1156	0.11290
151-200	0.07992	0.08619	**0.10778**	0.08642	0.09375	0.08577	0.09668
201-250	0.0	0.0	0.0	0.0	0.0	0.0	0.0
251-300	0.0	0.0	0.0	0.0	0.0	0.0	0.0
300+	0.11521	0.16210	0.12260	0.1752	**0.1786**	0.1680	0.16771
Combined	0.02169	0.02523	0.03419	0.03483	0.0366	**0.03733**	0.03285

Table 11: *NDCG@50 for Amazon-Cell-Sport where only one convolution layer is used.*

methods represent users and items as parameterized vectors and learn these parameters by reconstructing historical user-item interaction data. One of the earliest examples of this is Matrix factorization(MF) which maps users and items in a joint latent factor space with the same dimensionality and then models the user-item interactions as inner products in that space [9]. These representations of users and items are also commonly called embeddings. MF then uses the dot product of these to predict a user-item interaction. These methods do not require any domain knowledge because they only represent the users and items by their ID without any additional information. Newer methods also use neural networks to improve this model while still using the older format of embeddings. Examples of this are NCF [4] which replaces the inner product with a deep neural network and LRML[15] which uses neural networks to improve the learned embeddings. SVD++ [10] combines the neighborhood and the latent factor approach while also taking implicit feedback into account when calculating the predictions.

5.2. Graph based recommenders

Another research area that is relevant to our paper is Graph based recommenders.

will write about graph based recommenders

5.3. Extensions to LightGCN

There have been multiple extensions to LightGCN and work created inspired by LightGCN [13, 11, 17, 12]. One of them is LightGCN based Aspect-level Collaborative Filtering (LGC-ACF), which utilizes LightGCN and adds side information [13]. The input to this model is a user-item graph and a user-side information graph. These two graphs independently use the propagation method from LightGCN (as seen on Equation 6). For layer combination, LGC-ACF uses means as their aggregation function when combining the user embeddings created from the user-item graph and the user embeddings from the user-side information graph. This is also done with the item embedding [13]. Interest-aware Message-Passing GCN (IMP-GCN) is an alternative extension to LightGCN [11]. It alleviates the problem of GCNs being over-smoothed by creating clusters within the graphs so that not all nodes will end up being connected. The result of this method is that it can utilize a higher number of convolutions with good

performance compared to LightGCN [11]. Both of these methods could probably benefit from using a different layer combination method than just using means.

6. Future work

7. Conclusion

References

[1] Max Welling Rianne van den Berg Thomas N. Kipf. "Graph Convolutional Matrix Completion". In: *KDD workshop* (2017).

[2] Xavier Glorot and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks". In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. URL: http://proceedings.mlr.press/v9/glorot10a.html.

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

[4] Xiangnan He et al. "Neural Collaborative Filtering". In: *Proceedings of the 26th International Conference on World Wide Web*. WWW '17. Perth, Australia: International World Wide Web Conferences Steering Committee, 2017, 173–182. ISBN: 9781450349130. DOI: 10.1145/3038912.3052569. URL: https://doi.org/10.1145/3038912.3052569.

[5] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017.

[6] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017.

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

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

[9] Y. Koren, R. Bell, and C. Volinsky. "Matrix Factorization Techniques for Recommender Systems". In: *Computer* 42.8 (2009), pp. 30–37.

[10] Yehuda Koren. "Factorization Meets the Neighborhood: A Multifaceted Collaborative Filtering Model". In: *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '08. Las Vegas, Nevada, USA: Association for Computing Machinery, 2008, 426–434. ISBN: 9781605581934. DOI: 10.1145/1401890.1401944. URL: https://doi.org/10.1145/1401890.1401944.

[11] Fan Liu et al. "Interest-aware Message-Passing GCN for Recommendation". In: *CoRR* (2021).

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

[13] D. Mei, N. Huang, and X. Li. "Light Graph Convolutional Collaborative Filtering With Multi-Aspect Information". In: *IEEE Access* (2021).

[14] Steffen Rendle et al. "BPR: Bayesian Personalized Ranking from Implicit Feedback". In: UAI '09. 2009.

[15] Yi Tay, Luu Anh Tuan, and Siu Cheung Hui. "Latent Relational Metric Learning via Memory-based Attention for Collaborative Ranking". In: *Proceedings of the 2018 World Wide Web Conference on World Wide Web - WWW '18* (2018). DOI: 10.1145/3178876.3186154. URL: http://dx.doi.org/10.1145/3178876.3186154.

[16] 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).

[17] Wenhui Yu et al. "Self-propagation Graph Neural Network for Recommendation". In: *IEEE Transactions on Knowledge and Data Engineering* (2021).

Appendices

A. Changing α_k

This section contains the other results of removing α_k and removing the 0th embedding as described in **??**.

A.1. Removing α_k

Figure 13 shows the recall results for LightGCN and LightGCN-Ak1 on Yelp2020. Figure 14 and Figure 15 shows LightGCN and LightGCN-Ak1 on Amazon-Book.

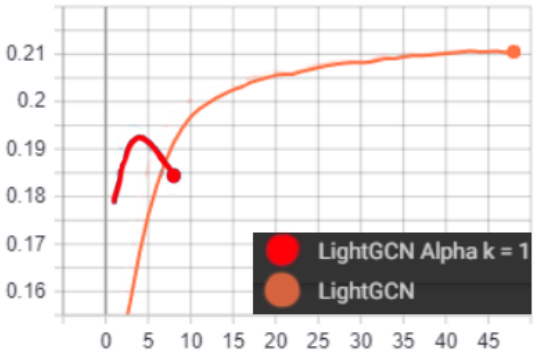

Figure 13: *Recall@50 of LightGCN and LightGCN-Ak1 on yelp2020*

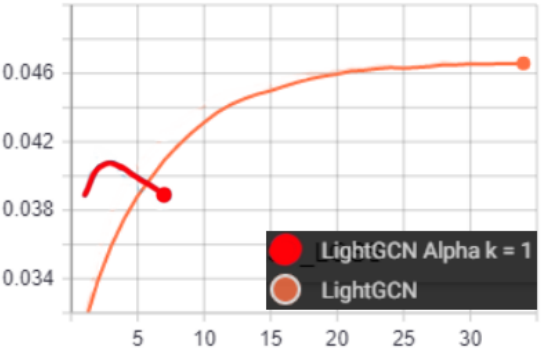

Figure 14: *NDCG@50 of LightGCN and LightGCN-Ak1 on amazon-book*

A.2. Removing embedding 0

On Table 12 and Table 13 the impact of removing $E^{(0)}$ can be seen. This also includes the impact within the different node degrees, where it in Amazon-Cell-Sport generally increases performance, but for Yelp2020 and Amazon-Book makes performance decrease.

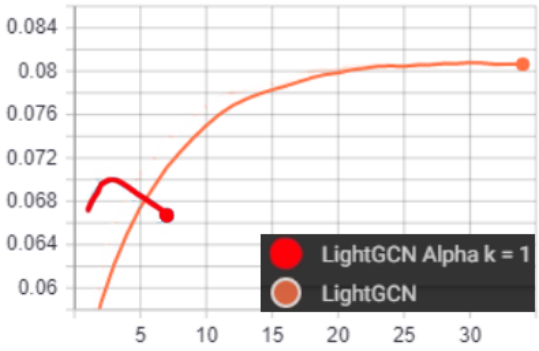

Figure 15: *Recall@50 of LightGCN and LightGCN-Ak1 on amazon-book*

B. Recall results for GCF ablation study

This section contains the recall results for the experiment described in Section 4.3. Additionally, some figures have been split up to only include the methods utilizing summation or concatenation to more easily compare the methods.

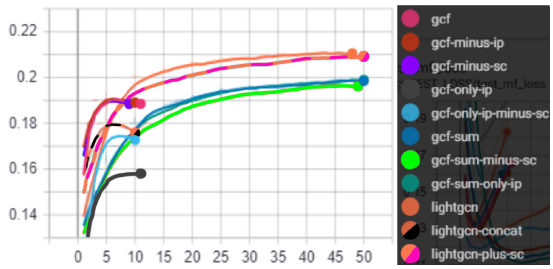

Figure 16: *Recall@50 on the Yelp2020 dataset.*

NDCG@50	Amazon-Cell-Sport		Yelp2020		Amazon-Book	
Method	5 con average	$E^{(0)}$ removed	5 con average	$E^{(0)}$ removed	3 con average	$E^{(0)}$ removed
6-10	0.01972	**0.02280**	**0.10095**	0.09961	0.0	0.0
11-15	**0.02984**	0.02908	**0.10408**	0.10154	0.0	0.0
16-20	**0.03701**	0.03631	**0.11451**	0.11107	**0.04929**	0.04727
21-25	**0.04438**	0.04278	**0.11790**	0.11299	**0.04831**	0.04731
26-30	0.04815	**0.04921**	**0.11909**	0.11805	**0.04778**	0.04625
31-35	0.07514	**0.07547**	**0.12833**	0.12237	**0.04825**	0.04769
36-40	0.05475	**0.05488**	**0.12711**	0.11891	**0.04493**	0.04473
41-45	0.07634	**0.07772**	**0.11638**	0.11260	**0.04301**	0.04220
46-50	0.10074	**0.10644**	**0.12537**	0.12101	0.04368	**0.04393**
51-60	0.08827	**0.09111**	**0.12459**	0.12090	**0.04210**	0.04066
61-70	0.07718	**0.09565**	**0.12377**	0.11745	0.03972	**0.04052**
71-80	0.05834	**0.06106**	**0.13007**	0.12483	0.03923	**0.04026**
81-90	0.07351	**0.07645**	**0.11042**	0.10217	0.03551	**0.03591**
91-100	**0.08968**	0.08920	**0.14010**	0.13954	0.03781	**0.03954**
101-150	**0.11290**	0.10939	**0.12676**	0.11947	0.03334	**0.03405**
151-200	0.09668	**0.10142**	**0.11889**	0.11728	0.02932	**0.03080**
201-250	0.0	0.0	**0.11999**	0.10884	0.03242	**0.03392**
251-300	0.0	0.0	0.21693	**0.22622**	0.03617	**0.03480**
301+	**0.16771**	0.16028	**0.23305**	0.20351	0.04130	**0.04380**
All nodes	0.03285	**0.03460**	**0.1089**	0.10641	**0.04668**	0.04470

Table 12: *Embedding 0 removed and the best performing methods.*

Recall@50	Amazon-Cell-Sport		Yelp2020		Amazon-Book	
Method	5 con average	$E^{(0)}$ removed	5 con average	$E^{(0)}$ removed	3 con average	$E^{(0)}$ removed
6-10	0.04706	**0.05126**	**0.23023**	0.22913	0.0	0.0
11-15	**0.06429**	0.06349	**0.22722**	0.22057	0.0	0.0
16-20	**0.07557**	0.07393	**0.22093**	0.21492	**0.09968**	0.09483
21-25	**0.08262**	0.07504	**0.21102**	0.20208	**0.09102**	0.08864
26-30	**0.08784**	0.08678	**0.19883**	0.19412	**0.08366**	0.08086
31-35	**0.10000**	0.09767	**0.20093**	0.19318	**0.08022**	0.07995
36-40	**0.08603**	0.08265	**0.19039**	0.17958	**0.07347**	0.07213
41-45	0.09644	**0.10171**	**0.16928**	0.16031	**0.06621**	0.06521
46-50	0.14301	**0.15499**	**0.17777**	0.17284	0.06414	**0.06438**
51-60	0.09665	**0.10110**	**0.16653**	0.16318	0.06002	**0.05816**
61-70	0.08252	**0.09962**	**0.15473**	0.14972	0.05359	**0.05446**
71-80	0.08231	**0.09158**	**0.16026**	0.15303	0.05036	**0.05139**
81-90	0.07068	**0.08068**	**0.12724**	0.12089	**0.04557**	0.04482
91-100	0.10312	**0.10375**	0.14825	**0.14837**	0.04576	**0.04648**
101-150	**0.12200**	0.11793	**0.13183**	0.12333	0.03746	**0.03794**
151-200	0.10173	**0.11455**	**0.11065**	0.10754	0.02945	**0.03103**
201-250	0.0	0.0	**0.10450**	0.09882	0.02762	**0.02864**
251-300	0.0	0.0	0.15278	**0.18056**	**0.02463**	0.02383
301+	**0.07792**	0.06494	**0.09083**	0.08452	0.01763	**0.01839**
All nodes	0.06451	**0.06726**	**0.2177**	0.21289	**0.08129**	0.07715

Table 13: *Embedding 0 removed and the best performing methods.*

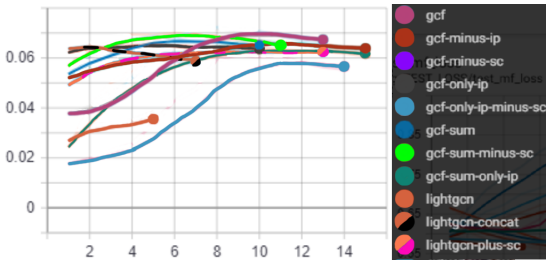

Figure 17: *Recall@50 on the Amazon-Sport-Cell dataset.*

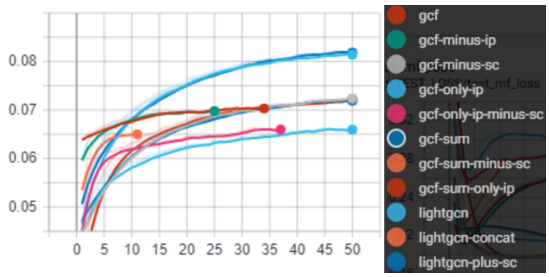

Figure 18: *Recall@50 on the Amazon-Book dataset.*

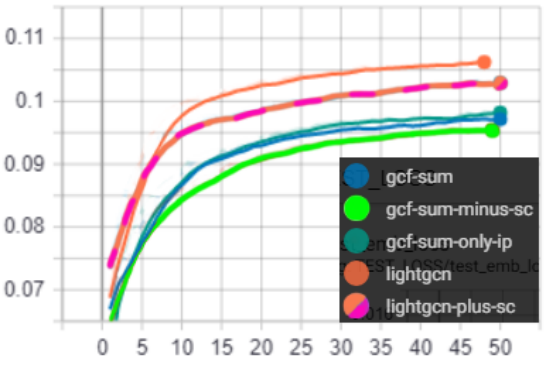

Figure 19: *NDCG@50 for the compared methods that utilize summation as layer combination on the Yelp2020 dataset.*

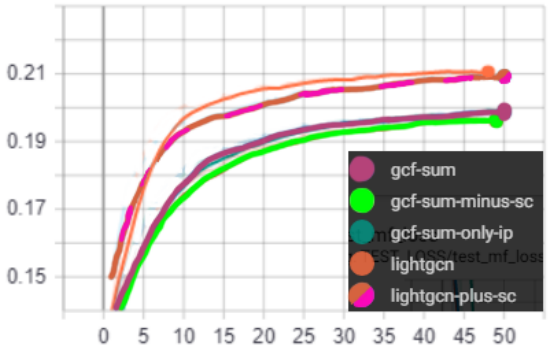

Figure 20: *Recall@50 on the compared methods that utilize summation as layer combination on the Yelp2020 dataset.*

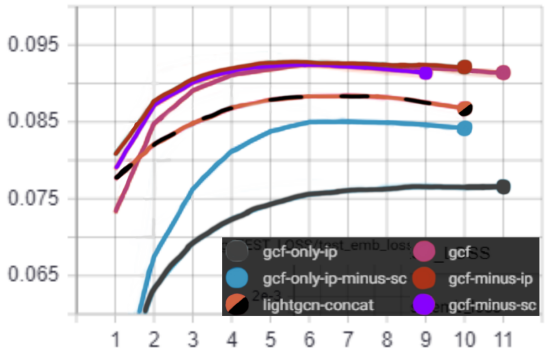

Figure 21: *NDCG@50 for the compared methods that utilize concatenation as layer combination on the Yelp2020 dataset.*

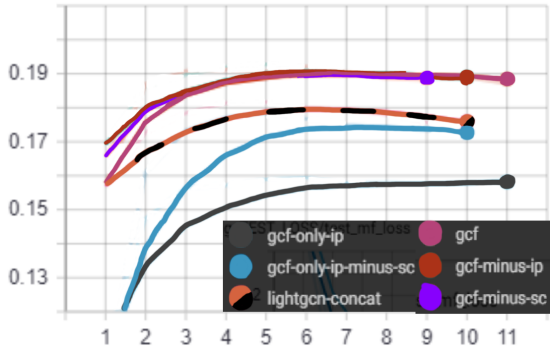

Figure 22: *Recall@50 for the compared methods that utilize concatenation as layer combination on the Yelp2020 dataset.*

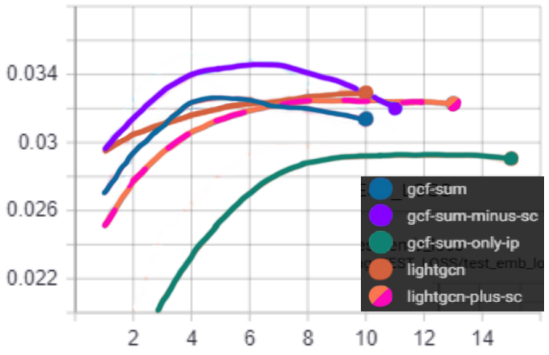

Figure 23: *NDCG@50 for the compared methods that utilize summation as layer combination on the Amazon-Cell-Sport dataset.*

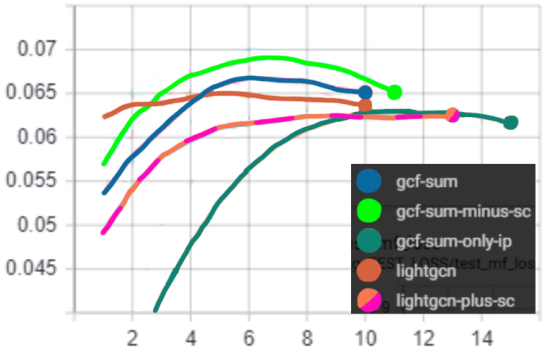

Figure 24: *Recall@50 on the compared methods that utilize summation as layer combination on the Amazon-Cell-Sport dataset.*

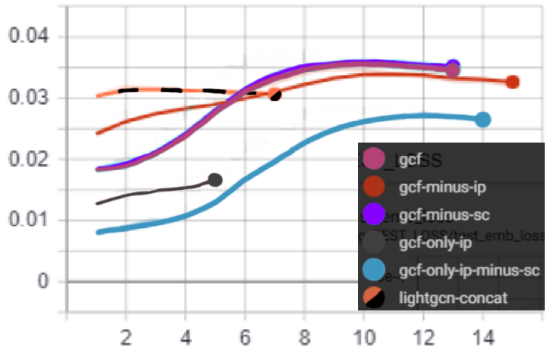

Figure 25: *NDCG@50 for the compared methods that utilize concatenation as layer combination on the Amazon-Cell-Sport dataset.*

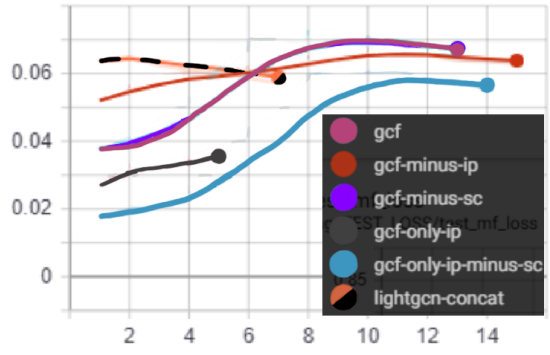

Figure 26: *Recall@50 for the compared methods that utilize concatenation as layer combination on the Amazon-Cell-Sport dataset.*

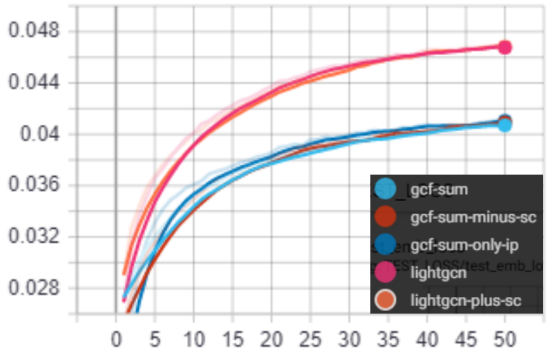

Figure 27: *NDCG@50 for the compared methods that utilize summation as layer combination on the Amazon-Book dataset.*

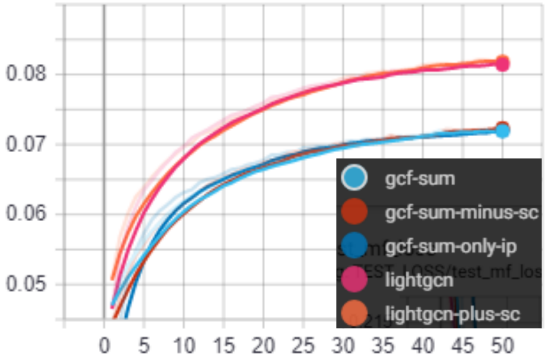

Figure 28: *Recall@50 on the compared methods that utilize summation as layer combination on the Amazon-Book dataset.*

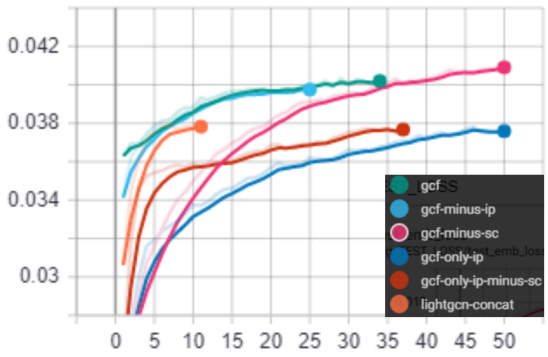

Figure 29: *NDCG@50 for the compared methods that utilize concatenation as layer combination on the Amazon-Book dataset.*

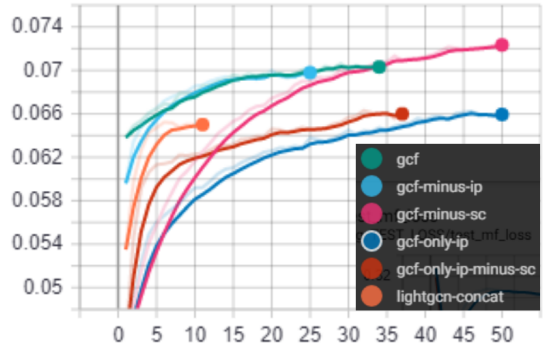

Figure 30: *Recall@50 for the compared methods that utilize concatenation as layer combination on the Amazon-Book dataset.*

C. One layer node degrees performance

This section contains the recall performance from the experiments conducted in Section 4.6.

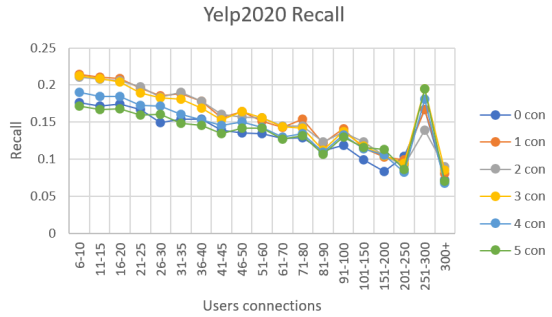

Figure 31: *Recall results for Yelp2020*

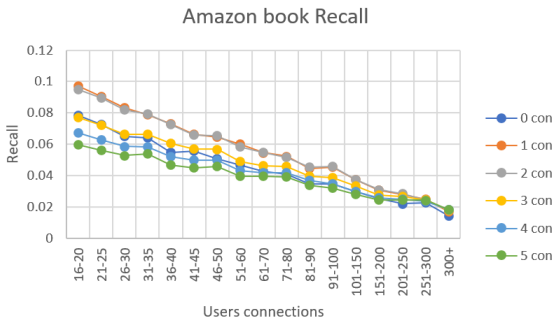

Figure 32: *Recall results for Amazon-Book*

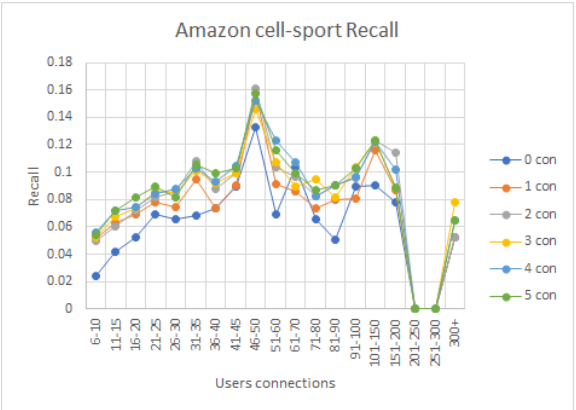

Figure 33: *Recall results for amazon-cell-sport*

Node degree	$E^{(0)}$	$E^{(1)}$	$E^{(2)}$	$E^{(3)}$	$E^{(4)}$	$E^{(5)}$	5 con
6-10	0.1766	0.2139	0.21017	0.2125	0.1905	0.1717	**0.23023**
11-15	0.1713	0.2107	0.20798	0.2087	0.1847	0.1672	**0.22722**
16-20	0.1740	0.2091	0.20607	0.2038	0.1843	0.1675	**0.22093**
21-25	0.1673	0.1957	0.19795	0.1895	0.1723	0.1599	**0.21102**
26-30	0.1489	0.1854	0.18368	0.1830	0.1712	0.1601	**0.19883**
31-35	0.1543	0.1884	0.18999	0.1807	0.1602	0.1486	**0.20093**
36-40	0.1541	0.1781	0.17817	0.1687	0.1528	0.1459	**0.19039**
41-45	0.1395	0.1562	0.16017	0.1534	0.1456	0.1347	**0.16928**
46-50	0.1354	0.1643	0.15675	0.1639	0.1501	0.1420	**0.17777**
51-60	0.1340	0.1509	0.15542	0.1554	0.1432	0.1420	**0.16653**
61-70	0.1279	0.143	0.14453	0.1440	0.1299	0.1274	**0.15473**
71-80	0.1288	0.1537	0.14475	0.1418	0.1347	0.1313	**0.16026**
81-90	0.1112	0.1214	0.12355	0.1119	0.1089	0.1068	**0.12724**
91-100	0.1182	0.1407	0.13647	0.1370	0.1325	0.1300	**0.14825**
101-150	0.09917	0.1153	0.12302	0.1174	0.1144	0.1155	**0.13183**
151-200	0.08331	0.1025	0.10502	0.1045	0.1053	**0.1132**	0.11065
201-250	0.1035	0.09954	0.09160	0.09430	0.08280	0.08620	**0.10450**
251-300	0.1667	0.1667	0.13889	**0.1944**	0.1806	**0.1944**	0.15278
300+	0.07180	0.0803	0.09024	0.08523	0.06765	0.06990	**0.09083**
All nodes	0.1674	0.2039	0.217	0.2001	0.01788	0.1643	**0.2177**

Table 14: *Recall@50 for Yelp2020*

Node degree	$E^{(0)}$	$E^{(1)}$	$E^{(2)}$	$E^{(3)}$	$E^{(4)}$	$E^{(5)}$	3 con
16-20	0.07856	0.09708	0.09492	0.07688	0.06712	0.05949	**0.09968**
21-25	0.07259	0.09056	0.0895	0.07192	0.06258	0.05611	**0.09102**
26-30	0.06510	0.08314	0.08181	0.06617	0.05848	0.05270	**0.08366**
31-35	0.06404	0.07888	0.07923	0.06632	0.05816	0.05369	**0.08022**
36-40	0.05481	0.07316	0.07235	0.06066	0.05222	0.04680	**0.07347**
41-45	0.05572	**0.06625**	0.06567	0.05713	0.04960	0.04508	0.06621
46-50	0.05072	0.06469	**0.06536**	0.05670	0.04990	0.04589	0.06414
51-60	0.04664	**0.06012**	0.05824	0.04883	0.04301	0.03945	0.06002
61-70	0.04275	0.05449	**0.05477**	0.04623	0.04170	0.03952	0.05359
71-80	0.04068	**0.05224**	0.05174	0.04564	0.04162	0.03902	0.05036
81-90	0.03445	0.04458	0.04545	0.03973	0.03678	0.03381	**0.04557**
91-100	0.03458	0.04517	0.04575	0.03872	0.03462	0.03189	**0.04576**
101-150	0.02960	0.03731	0.03732	0.03326	0.02979	0.02815	**0.03746**
151-200	0.02537	0.03047	**0.03086**	0.02760	0.02582	0.02442	0.02945
201-250	0.02193	0.02781	**0.02824**	0.02672	0.02493	0.02436	0.02762
251-300	0.02276	**0.02480**	0.02437	0.02438	0.02351	0.02418	0.02463
301+	0.01397	0.01689	0.01768	0.01783	**0.01797**	0.01796	0.01763
All nodes	0.06373	0.079	0.07755	0.06412	0.05607	0.05022	**0.08129**

Table 15: *Recall@50 for Amazon-Book where only one convolution layer is used.*

Node degree	$E^{(0)}$	$E^{(1)}$	$E^{(2)}$	$E^{(3)}$	$E^{(4)}$	$E^{(5)}$	5 con
6-10	0.02437	0.05126	0.05000	0.05210	**0.05546**	0.05420	0.04706
11-15	0.04286	0.06305	0.06067	0.06764	0.07143	**0.07213**	0.06429
16-20	0.05224	0.06870	0.07192	0.07338	0.07415	**0.08123**	0.07557
21-25	0.06957	0.07769	0.08148	0.08611	0.08430	**0.08944**	0.08262
26-30	0.06574	0.07475	0.08599	0.08492	**0.08790**	0.08176	0.08784
31-35	0.06837	0.09441	**0.10828**	0.1011	0.1025	0.1050	0.10000
36-40	0.07322	0.07343	0.08794	0.09234	0.09341	**0.09957**	0.08603
41-45	0.08924	0.09060	0.09939	0.09934	**0.1045**	0.1030	0.09644
46-50	0.13245	0.15787	**0.16085**	0.1459	0.1518	0.1577	0.14301
51-60	0.06920	0.09088	0.10339	0.1075	**0.1227**	0.1163	0.09665
61-70	0.10338	0.08628	0.09633	0.08954	**0.1069**	0.09962	0.08252
71-80	0.06541	0.07399	0.08256	**0.09443**	0.08206	0.08658	0.08231
81-90	0.05063	0.07967	**0.09073**	0.08120	**0.09073**	0.09020	0.07068
91-100	0.08926	0.08070	0.09555	**0.1038**	0.09618	0.1031	0.10312
101-150	0.08992	0.11622	**0.12334**	0.1231	0.1226	0.1228	0.12200
151-200	0.07804	0.08696	**0.11455**	0.08891	0.1017	0.08891	0.10173
201-250	0.0	0.0	0.0	0.0	0.0	0.0	0.0
251-300	0.0	0.0	0.0	0.0	0.0	0.0	0.0
300+	0.05195	0.05195	0.05195	**0.07792**	0.06494	0.06494	**0.07792**
All nodes	0.04447	0.04859	0.06809	0.06972	0.7377	0.07318	0.06451

Table 16: *Recall@50 for Amazon-Cell-Sport*

D. LightGCN results

This section shows the performance of LightGCN on Yelp2020, Amazon-Book and Amazon-Cell-Sport with one to five convolutions. The performance of different node degrees is also evaluated. This was conducted to gain an understanding of how well the different node degrees perform in LightGCN with its standard layer combination.

NDCG@50	1 con	2 con	3 con	4 con	5 con
6-10	0.08704	0.09136	0.09779	0.10049	**0.10095**
11-15	0.09078	0.09444	0.10038	0.10316	**0.10408**
16-20	0.10259	0.10575	0.11117	0.11322	**0.11451**
21-25	0.10777	0.10892	0.11442	0.11546	**0.11790**
26-30	0.10708	0.10933	0.11892	**0.11943**	0.11909
31-35	0.11726	0.12151	0.12698	0.12812	**0.12833**
36-40	0.11146	0.11764	0.12093	0.12457	**0.12711**
41-45	0.10448	0.11056	0.11495	**0.11795**	0.11638
46-50	0.10790	0.11447	0.12181	0.12132	**0.12537**
51-60	0.10763	0.11560	0.12132	0.12431	**0.12459**
61-70	0.10582	0.11338	0.11768	0.11901	**0.12377**
71-80	0.12429	0.12072	0.12936	0.12837	**0.13007**
81-90	0.10092	0.10477	**0.11046**	0.11017	0.11042
91-100	0.12217	0.13189	0.13780	**0.14200**	0.14010
101-150	0.11807	0.11731	0.12361	0.12482	**0.12676**
151-200	0.10340	0.11441	0.11854	**0.11968**	0.11889
201-250	0.13015	0.11866	0.12730	**0.13091**	0.11999
251-300	0.20810	0.17787	0.19453	**0.22210**	0.21693
301+	0.10245	0.21585	0.20725	0.22121	**0.23305**
All nodes	0.0969	0.1008	0.1064	0.1084	**0.1089**

Table 17: *NDCG@50 for Yelp2020 with a different number of convolutions*

Recall@50	1 con	2 con	3 con	4 con	5 con
6-10	0.20318	0.21017	0.22301	**0.23146**	0.23023
11-15	0.20017	0.20798	0.21843	0.22592	**0.22722**
16-20	0.19930	0.20607	0.21462	0.21764	**0.22093**
21-25	0.19296	0.19795	0.20569	0.20589	**0.21102**
26-30	0.17911	0.18368	0.19649	0.19852	**0.19883**
31-35	0.17874	0.18999	0.19433	0.20008	**0.20093**
36-40	0.16906	0.17817	0.18199	0.18580	**0.19039**
41-45	0.15245	0.16017	0.16653	0.16778	**0.16928**
46-50	0.15517	0.15675	0.17168	0.17382	**0.17777**
51-60	0.14891	0.15542	0.16334	**0.16787**	0.16653
61-70	0.13905	0.14453	0.15074	0.15043	**0.15473**
71-80	0.15146	0.14475	0.15470	0.15043	**0.16026**
81-90	0.12426	0.12355	0.13509	**0.15796**	0.12724
91-100	0.12518	0.13647	0.14130	**0.14878**	0.14825
101-150	0.12217	0.12302	0.12738	0.13045	**0.13183**
151-200	0.09970	0.10502	0.10904	**0.11410**	0.11065
201-250	0.09952	0.09160	0.10636	**0.11138**	0.10450
251-300	0.16667	0.13889	**0.15278**	**0.15278**	**0.15278**
300+	0.07625	0.09024	0.07919	0.08636	**0.09083**
All nodes	0.1955	0.2015	0.2106	0.2157	**0.2177**

Table 18: *Recall@50 for Yelp2020 with a different number of convolutions*

NDCG@50	1 con	2 con	3 con	4 con	5 con
16-20	0.04677	**0.04981**	0.04929	0.04726	0.04774
21-25	0.04604	**0.04950**	0.04831	0.04714	0.04767
26-30	0.04466	0.04776	**0.04778**	0.04585	0.04632
31-35	0.04617	0.04823	**0.04825**	0.04742	0.04753
36-40	0.04124	0.04385	**0.04493**	0.04395	0.04360
41-45	0.04034	**0.04341**	0.04301	0.04155	0.04249
46-50	0.04083	0.04196	**0.04368**	0.04336	0.04330
51-60	0.04000	0.04200	**0.04210**	0.04077	0.04101
61-70	0.03645	0.03943	**0.03972**	0.03928	0.03846
71-80	0.03374	0.03730	**0.03923**	0.03788	0.03817
81-90	0.03292	0.03463	0.03551	**0.03554**	0.03498
91-100	0.03526	0.03658	**0.03781**	0.03675	0.03603
101-150	0.03062	0.03315	**0.03334**	0.03284	0.03305
151-200	0.02765	0.02841	**0.02932**	0.02879	0.02942
201-250	0.02921	0.03217	0.03242	**0.03317**	0.03272
251-300	0.03205	0.03414	**0.03617**	0.03607	0.03602
300+	0.03643	0.03707	**0.04130**	0.04110	0.04207
All nodes	0.0427	0.0463	**0.04668**	0.04617	0.04515

Table 19: *NDCG@50 for Amazon-Book with a different number of convolutions*

Recall@50	1 con	2 con	3 con	4 con	5 con
16-20	0.09394	**0.10047**	0.09968	0.09623	0.09723
21-25	0.08550	**0.09182**	0.09102	0.08918	0.09038
26-30	0.07852	**0.08425**	0.08366	0.08139	0.08185
31-35	0.07506	0.08017	**0.08022**	0.07843	0.07733
36-40	0.06697	0.07160	**0.07347**	0.07117	0.07037
41-45	0.06233	**0.06649**	0.06621	0.06359	0.06514
46-50	0.06091	0.06240	0.06414	**0.06442**	0.06408
51-60	0.05604	**0.06050**	0.06002	0.05836	0.05861
61-70	0.04949	0.05242	**0.05359**	0.05318	0.05278
71-80	0.04396	0.04869	**0.05036**	0.04861	0.04929
81-90	0.04015	0.04286	**0.04557**	0.04487	0.04459
91-100	0.04202	0.04338	**0.04576**	0.04429	0.04359
101-150	0.03416	0.03661	**0.03746**	0.03711	0.03731
151-200	0.02772	0.02800	**0.02945**	0.02910	0.02993
201-250	0.02487	0.02713	0.02762	**0.02782**	0.02757
251-300	0.02279	0.02376	0.02463	0.02485	**0.02496**
300+	0.01542	0.01594	0.01763	0.01735	**0.01790**
All nodes	0.07408	0.08055	**0.08129**	0.08033	0.07861

Table 20: *Recall@50 for Amazon-Book with a different number of convolutions*

NDCG@50	1 con	2 con	3 con	4 con	5 con
6-10	0.01831	0.02004	**0.02250**	0.02140	0.01972
11-15	0.02448	0.02729	0.02713	0.02884	**0.02984**
16-20	0.02610	0.03200	0.03603	0.03531	**0.03701**
21-25	0.03618	0.04034	0.04240	0.04318	**0.04438**
26-30	0.04344	0.04152	0.04643	0.04587	**0.04815**
31-35	0.06181	0.06487	0.07105	0.06915	**0.07514**
36-40	0.05080	0.05420	0.05093	**0.05490**	0.05475
41-45	0.07061	**0.07714**	0.07564	0.07637	0.07634
46-50	0.10666	0.10287	0.10973	**0.11381**	0.10074
51-60	0.08322	**0.08399**	0.08040	0.08078	0.08827
61-70	**0.09955**	0.08355	0.08448	0.08521	0.07718
71-80	0.05834	0.05057	**0.06090**	0.06017	0.05834
81-90	0.03095	0.05879	0.05654	0.06116	**0.07351**
91-100	0.07926	0.08009	**0.10496**	0.09765	0.08968
101-150	0.09431	0.10686	0.10512	0.10775	**0.11290**
151-200	0.04864	**0.09700**	0.09278	0.08609	0.09668
301+	0.06569	0.15789	0.17848	**0.18099**	0.16771
All nodes	0.02804	0.03132	0.03237	0.03253	**0.03285**

Table 21: *NDCG@50 for Amazon-Cell-Sport with a different number of convolutions*

Recall@50	1 con	2 con	3 con	4 con	5 con
6-10	0.04496	0.04538	**0.05084**	**0.05084**	0.04706
11-15	0.05229	0.05847	0.05644	0.06393	**0.06429**
16-20	0.05296	0.06519	0.07155	0.06909	**0.07557**
21-25	0.07538	0.07572	0.07740	0.08018	**0.08262**
26-30	0.07759	0.07310	0.08362	0.08249	**0.08784**
31-35	0.08349	0.09110	0.09457	0.09658	**0.10000**
36-40	0.07398	**0.09017**	0.08287	0.08990	0.08603
41-45	0.10083	**0.10321**	0.09576	0.09012	0.09644
46-50	0.15792	0.14027	**0.16305**	0.15792	0.14301
51-60	0.08950	**0.10754**	0.10162	0.09975	0.09665
61-70	**0.10360**	0.08954	0.09048	0.08979	0.08252
71-80	0.08158	0.07376	**0.09111**	0.07921	0.08231
81-90	0.04110	**0.07068**	**0.07068**	**0.07068**	**0.07068**
91-100	0.08103	0.08103	**0.01182**	0.10343	0.10312
101-150	0.09462	0.11410	0.11453	0.12082	**0.12200**
151-200	0.05435	0.09783	0.09978	0.08891	**0.10173**
300+	0.05195	0.06494	0.06494	**0.09091**	0.07792
All nodes	0.05503	0.06133	0.06447	0.06394	**0.06451**

Table 22: *Recall@50 for Amazon-Cell-Sport with a different number of convolutions*

D.1. Adjusted layer combination

E. LightGCN - One weight added

The core difference between NGCF and LightGCN is that LightGCN does not utilize learnable weight matrices and does not utlize an activation function [3, 16]. Our intuition behind adding a learnable weight matrix would be that we could learn how to perform the optimal layer combination, as there are learnable weights for each layers. The equation can be seen on Equation 16 Therefore we experimented with three different layer combinations: sum (Equation 17), average (Equation 18) and concatenation (Equation 19). As can be seen on Table 25 the results were quite poor compared to "autoref this to the baseline methods."

$$E^{(k+1)} = \sum_{i \in \mathcal{N}_u} \frac{1}{\sqrt{|\mathcal{N}_u||\mathcal{N}_i|}} \left(\mathbf{e}_i^{(k)} \mathbf{W}^{(k)} \right) \tag{16}$$

$$\mathbf{e}_u = \sum_{k=0}^{K} \mathbf{e}_u^{(k)}, \tag{17}$$

$$\mathbf{e}_u = \sum_{k=0}^{K} \frac{1}{k} \mathbf{e}_u^{(k)}, \tag{18}$$

$$\mathbf{e}_u = \mathbf{e}_u^{(0)} || \cdots || \mathbf{e}_u^{(K)}, \tag{19}$$

E.1. Individual layer performance

NDCG@50	Amazon-Cell-Sport		Yelp2020		Amazon-Book	
Method	5 con average	Aggressive split	5 con average	Aggressive split	3 con average	Aggressive split
6-10	0.01972	0.02311	0.10095		0.0	
11-15	0.02984	0.03009	0.10408		0.0	
16-20	0.03701	0.03567	0.11451		0.04929	
21-25	0.04438	0.04658	0.11790		0.04831	
26-30	0.04815	0.04567	0.11909		0.04778	
31-35	0.07514	0.07264	0.12833		0.04825	
36-40	0.05475	0.06066	0.12711		0.04493	
41-45	0.07634	0.08189	0.11638		0.04301	
46-50	0.10074	0.09847	0.12537		0.04368	
51-60	0.08827	0.09507	0.12459		0.04210	
61-70	0.07718	0.08675	0.12377		0.03972	
71-80	0.05834	0.06132	0.13007		0.03923	
81-90	0.07351	0.07105	0.11042		0.03551	
91-100	0.08968	0.09289	0.14010		0.03781	
101-150	0.11290	0.12259	0.12676		0.03334	
151-200	0.09668	0.07744	0.11889		0.02932	
201-250	0.0	0.0	0.11999		0.03242	
251-300	0.0	0.0	0.21693		0.03617	
301+	0.16771	0.17331	0.23305		0.04130	
All nodes	0.03285	0.03515	0.1089		0.04668	

Table 23: *Adjusted layer combination, where it was used within each node range.*

Recall@50	Amazon-Cell-Sport		Yelp2020		Amazon-Book	
Method	5 con average	Aggressive split	5 con average	Aggressive split	3 con average	Aggressive split
6-10	0.04706	0.05378	0.23023		0.0	
11-15	0.06429	0.06966	0.22722		0.0	
16-20	0.07557	0.07333	0.22093		0.09968	
21-25	0.08262	0.08910	0.21102		0.09102	
26-30	0.08784	0.07973	0.19883		0.08366	
31-35	0.10000	0.09876	0.20093		0.08022	
36-40	0.08603	0.09450	0.19039		0.07347	
41-45	0.09644	0.11017	0.16928		0.06621	
46-50	0.14301	0.13470	0.17777		0.06414	
51-60	0.09665	0.11614	0.16653		0.06002	
61-70	0.08252	0.08625	0.15473		0.05359	
71-80	0.08231	0.09063	0.16026		0.05036	
81-90	0.07068	0.08120	0.12724		0.04557	
91-100	0.10312	0.11007	0.14825		0.04576	
101-150	0.12200	0.13196	0.13183		0.03746	
151-200	0.10173	0.08696	0.11065		0.02945	
201-250	0.0		0.10450		0.02762	
251-300	0.0		0.15278		0.02463	
301+	0.07792	0.06494	0.09083		0.01763	
All nodes	0.06451	0.07117	0.2177		0.08129	

Table 24: *Adjusted layer combination, where it was used within each node range.*

	Amazon-Cell-Sport			Yelp2020			Amazon-Book		
Layer combination	Concat	Sum	Mean	Concat	Sum	Mean	Concat	Sum	Mean
NDCG@50	**0.02381**	0.01867	0.02075	0.03144	0.07482	**0.08514**	0.03144	0.03376	**0.03624**
Recall@50	**0.04777**	0.03910	0.04241	0.05345	0.15509	**0.17398**	0.05345	0.05828	**0.06197**

Table 25: *Results for LightGCN with different layer combinations, where one weighted is added to the embedding propagation*

NDCG@50	$\mathbf{e}^{(0)}$	$\mathbf{e}^{(1)}$	$\mathbf{e}^{(2)}$	$\mathbf{e}^{(3)}$	$\mathbf{e}^{(4)}$	$\mathbf{e}^{(5)}$
Amazon-Book	0.03669	**0.0458**	0.04487	0.0372	0.03247	0.02923
Yelp2020	0.08177	0.1019	**0.1086**	0.09956	0.08863	0.0819
Amazon-Cell-Sport	0.02169	0.02523	0.03419	0.03483	0.0366	**0.03733**
Amazon-Cloth-Sport	0.03092	0.06054	**0.06392**	0.05979	0.05928	0.05208
Amazon-Cell-Electronic	0.03211	0.04403	0.05204	**0.05422**	0.05331	0.05158
Amazon-Cloth-Electronic	0.00659	0.01429	0.01688	0.01915	0.02005	**0.02074**

Table 26: *NDCG@50 results for all datasets utilizing only 1 layer in LightGCN*

Recall@50	$\mathbf{e}^{(0)}$	$\mathbf{e}^{(1)}$	$\mathbf{e}^{(2)}$	$\mathbf{e}^{(3)}$	$\mathbf{e}^{(4)}$	$\mathbf{e}^{(5)}$
Amazon-Book	0.06373	**0.079**	0.07755	0.06412	0.05607	0.05022
Yelp2020	0.1674	0.2039	**0.217**	0.2001	0.01788	0.1643
Amazon-Cell-Sport	0.04447	0.04859	0.06809	0.06972	**0.7377**	0.07318
Amazon-Cloth-Sport	0.05190	0.09885	**0.10541**	0.10066	0.09930	0.09109
Amazon-Cell-Electronic	0.04706	0.06518	0.07587	**0.07909**	0.07623	0.07584
Amazon-Cloth-Electronic	0.01324	0.02724	0.03223	0.03721	0.03876	**0.04061**

Table 27: *Recall@50 results for all datasets utilizing only 1 layer in LightGCN*