

# Experimental Design - Graph Trend Filtering Networks for Recommendation

Teodor Chakarov 12141198 Aleksandar Manov 12143825 Aymeric Hollaus 0158903  
Noah Watzal 01633746

January 31, 2023

## 1 Introduction

For the second exercise of Experimental Design, we had to choose three research papers from different conferences between the 2021 and 2022 (e.g. SIGIR, CIKM, ECIR, RecSys). We have chosen for the following papers:

- GraFN: Semi-Supervised Node Classification on Graph with Few Labels via Non-Parametric Distribution Assignment
- Which Discriminator for Cooperative Text Generation?
- Graph Trend Filtering Networks for Recommendation

We choose the research paper [Graph Trend Filtering Networks for Recommendation](#) because we found this topic interesting, the paper was understandable for us and also there was a [GitHub repository](#) provided with the source, leading us to believe that reproducibility would be easy.

This paper expanded on recommender systems and offered a novel solution for this field. The key role of recommender systems is to predict whether users are likely to interact with items (e.g. products, songs, movies, etc.) based on their previous interactions, including clicks, add-to-cart, purchases, and so on. The research of the author also briefly went through what collaborative filtering (CF) techniques are. These techniques are developed to model user-item interactions on historical view, assuming that users who behave similarly are likely to have similar preferences towards items. The paper utilized this with neural networks to create neural collaborative filtering specifically their version of graph neural networks (GNNs), which is graph trend filtering networks for recommendations (GTN).

### 1.1 The stated problem

Despite the success of modeling user-item relationships, the paper elaborated on why it is insufficient to pinpoint the heterogeneous reliability of interactions among instances in recommender systems. The key reason is that the majority of existing "deep" recommender systems treat all interactions equally. This implies that for example click-bait and not intended clicks will be equally treated as intended user clicks, which we can all agree on to be a bad equivalence. Therefore, the authors opted to design a new collaborative filtering method that adaptively propagates the embedding in recommender systems, which can lead to more accurate and robust recommendations.

To better exploit these user-item interactions, graph collaborative filtering models such as NGCF and LightGCN, which this papers' solution is based upon, propose to propagate the user embedding and item embedding according to the user-item interaction through the propagation itself.

### 1.2 Suggested solution

Despite successful uses, heterogeneous reliability of interactions among instances in recommender systems is insufficient to discover the heterogeneous reliability and has shortcomings, as already mentioned above. Therefore, their novel idea is a new collaborative filtering method that adaptively propagates

$$\arg \min_{\mathbf{E} \in \mathbb{R}^{(n+m) \times d}} \frac{1}{2} \|\mathbf{E} - \mathbf{E}_{\text{in}}\|_F^2 + \lambda \|\tilde{\Delta} \mathbf{E}\|_1, \quad (7)$$

where  $\mathbf{E}_{\text{in}}$  contains the initial embeddings of items and users.  $\tilde{\Delta} = \Delta \mathbf{D}^{-\frac{1}{2}}$  is the incident matrix normalized by the square root of node degrees<sup>2</sup>.  $\lambda$  is a hyper-parameter to control the strength of the embedding smoothness. The first term preserves the proximity with the initial embedding of users and items, and the second term imposes embedding smoothness over the interactions  $\mathcal{E}$  since

Figure 1: Embedding smoothness objective for user-item graph in recommendations

the embedding in recommender systems, which can lead to more accurate and robust recommendations. Their Graph Trend filtering Networks (GTN) captures and learns the adaptive importance of the interactions in recommender systems. Additionally the proposed embedding smoothness objective for user-item graph in recommendations is convex so that local minima is equivalent of the global minimum. See Figure 1, there one can see the mentioned embedding smoothness objective, alongside an elaboration of it from the paper.

The paper compared GTN with 7 other models, one of them was the earlier mentioned LightGCN. LightGCN is or better put was the state-of-the-art model which performed better than the other 6 models, besides the newly suggested GTN which barely performed better than LightGCN. Their performances were tested on large and established datasets in the recommendation field, which are: Gowalla, Yelp2018, Amazon-Book and LastFM.

## 2 Strategy for experiment reproduction

Naturally our goal was to reproduce said experiment, or at the very least run their model and go on from there. In order to manage to reproduce the stated results, we started by downloading the code from the GitHub repository that was provided: <https://github.com/wenqifan03/GTN-SIGIR2022>. Dependencies were almost very well documented besides a few intricacies depending on what packets you already installed in your environment. We mainly worked in local *.venv* environment (each of us having 6GB of VRAM GPUs) and/or Google Colab. Even in the fresh environment, dependencies were hard to manage, the authors did not have a *requirements.txt* file, and only documented some packages in their *README*. After tinkering around we discovered a matching pair of *torch* 12 and *cuda* 10.2 and the according *torch-geometric* library. After we finally managed to get the dependencies working, we had to tweak the *world.py* script so that our GPUs could be recognized as the correct device, see Figure 2. We then executed the base commands:

```
$ cd code
$ python run_main.py --dataset 'gowalla' --lambda2 4.0
```

Running the script as such the model should be initialized with the default properties, seeds, model size, batch size, learning rate, normalization coefficients and so on. Even though this should have worked, the four of us were not able to reproduce it on our local devices and switched back to Google Colab since it provides free to use GPU processing power with CUDA acceleration for neural network training. Even there, we ran into a dreaded ‘tcmalloc: large alloc’, see Figure 3, error which stopped the models in its tracks as you will see in the screenshot below, in the **Difficulties** section.

## 3 Difficulties

The difficulties we saw in our process of experiment reproduction were mainly the following:

- Difficulties setting up the libraries: Firstly as already mentioned above the *requirements.txt* was missing from the source code. The requirements were described in the *README.md* file. The

```
File "C:\Users\user\AppData\Local\Programs\Python\Python38-64\env\lib\site-packages\torch\cuda\_init_.py", line 314, in set_device
    torch._C._cuda_setDevice(device)
RuntimeError: CUDA error: invalid device ordinal
CUDA kernel errors might be asynchronously reported at some other API call, so the stacktrace below might be incorrect.
For debugging consider passing CUDA_LAUNCH_BLOCKING=1.
```

Figure 2: Error with device config

```
>>>SEED: 2020
loading ../data/gowalla
810128 interactions for training
217242 interactions for testing
gowalla Sparsity : 0.0008396216228570436
gowalla is ready to go
=====Config=====
{'A_n_fold': 100,
 'A_split': False,
 'K': 3,
 'args': Namespace(K=3, a_fold=100, alpha=0.3, alpha1=0.25, alpha2=0.25, avg=0, beta=0.5, bpr_batch=2048, comment='gtm', dataset='gowalla', debug=True, decay=
 'bigdata': False,
 'bpr_batch_size': 2048,
 'dataset': 'gowalla',
 'decay': 0.0001,
 'dropout': 0,
 'epochs': 1000,
 'keep_prob': 0.6,
 'lambda2': 4.0,
 'latent_dim_rec': 256,
 'lr': 0.001,
 'multicore': 1,
 'pretrain': 0,
 'test_u_batch_size': 100}
cores for test: 1
comment: gtm
tensorboard: 0
LOAD: 0
weight path: ./checkpoints
Test Topks: [20]
using bpr loss
=====end=====
to gowalla distribution initializer
loading adjacency matrix
generating adjacency matrix
tcmalloc: large alloc 9788891136 bytes == 0x7f176ea3e000 @ 0x7f1a4dcad001 0x7f19f935322f 0x7f19f93ab68b 0x7f19f93ac4f7 0x7f19f944e913 0x5aae14 0x49abe4 0x4fc
tcmalloc: large alloc 4894449664 bytes == 0x7f164a688000 @ 0x7f1a4dcab1e7 0x7f19f935314e 0x7f19f93af166 0x7f19f93af549 0x7f19f93b0fe8 0x7f19f93b16f9 0x7f19f93
~C
```

Figure 3: tcmalloc: large alloc

other struggles regarding version of the libraries were already mentioned as well, so we will not go into detail with this.

- Understanding the code. The code was split up in multiple sections since they were all scripts. One responsible for loading, parsing, modeling, propagation, etc. This abstraction/split was fairly helpful for us to guide us through their experimental process programmatically, nonetheless understanding it wasn't as easy as we made it out to be. We can confidently say that we understood the base principles and the majority of functions, but some variable assignments or library function usages are still a bit blurry to us. Lastly we noticed that for the training phase, they used negative sampling and interestingly enough they conducted their sampling with a *.cpp* file and python file. Their interaction was conducted through *pybind11* which allows operability between *C++* and *Python*.
- Difficulties executing the code and not enough hardware power. Again, we based ourselves on the provided *GitHub* repository and their base command: **python run\_main.py --dataset 'gowalla' --lambda2 4.0**. The issues we had were touched upon above but else the command on it's own ran. Our limitation resulted in memory allocation, which we heavily suspect the VRAM being the bottleneck. The requirement of at least 9GB of RAM space for initialization was also a bottleneck, but only for Google Colab. In order to bypass this problem we tried to reduce the model size by reducing the argument K from 3 by default to 1. We still got this error. This type of error occur as we try to initialize the model, we cant reach the training phase. So reducing the batch size and n folds didn't help. It should be noted that neither in *GitHub*, nor in their paper, was it written what devices they were using, or what setup, to run these experiments. This would have been beneficial to know before selecting the paper.

## 4 Alternative & Results

Since there we had no success reproducing the numbers and any results of the paper, we opted to at least try to run one of the baseline models and compare these results with the given paper. We chose the next best model, which was their baseline model, LightGCN, as mentioned in the the paper

```
python main.py --decay=1e-4 --lr=0.001 --layer=3 --seed=2020 --dataset="gowalla" --topks="[20]" --recdim=64

cpp extension not loaded
>>SEED: 2020
Loading ./dataset/gowalla
810128 interactions for training
217242 interactions for testing
gowalla sparsity: 0.0008790216228570436
gowalla is ready to go
=====config=====
{'A_n_fold': 100,
 'A_split': False,
 'bigdata': False,
 'bpr_batch_size': 2048,
 'decay': 0.0001,
 'dropout': 0,
 'keep_prob': 0.6,
 'latent_dim_rec': 64,
 'lightgc_n_layers': 3,
 'lr': 0.001,
 'multicore': 0,
 'pretrain': 0,
 'test_u_batch_size': 100}
cores for test: 1
comment: lgn
tensorboard: 1
load: 0
weight path: ./checkpoints
Test Topks: [20]
using bpr loss
=====end=====

[TEST]
{'precision': array([0.00018755]), 'recall': array([0.00053749]), 'ndcg': array([0.00040836])}
EPOCH[1/1000] loss0.545-[Sample:11.07]
EPOCH[2/1000] loss0.240-[Sample:12.63]
EPOCH[3/1000] loss0.163-[Sample:11.21]
EPOCH[4/1000] loss0.131-[Sample:10.73]
EPOCH[5/1000] loss0.112-[Sample:10.92]
EPOCH[6/1000] loss0.099-[Sample:11.40]
EPOCH[7/1000] loss0.090-[Sample:11.58]
EPOCH[8/1000] loss0.084-[Sample:10.61]
EPOCH[9/1000] loss0.078-[Sample:12.63]
EPOCH[10/1000] loss0.074-[Sample:10.53]
[TEST]
{'precision': array([0.03665852]), 'recall': array([0.12015337]), 'ndcg': array([0.10059577])}
EPOCH[11/1000] loss0.071-[Sample:10.72]
EPOCH[12/1000] loss0.068-[Sample:10.62]
EPOCH[13/1000] loss0.065-[Sample:10.79]
EPOCH[14/1000] loss0.064-[Sample:10.73]
EPOCH[15/1000] loss0.061-[Sample:11.20]
EPOCH[16/1000] loss0.059-[Sample:11.81]
EPOCH[17/1000] loss0.057-[Sample:11.21]
EPOCH[18/1000] loss0.055-[Sample:11.91]
EPOCH[19/1000] loss0.054-[Sample:11.12]
EPOCH[20/1000] loss0.052-[Sample:11.72]
```

Figure 4: Results of LightGCN.

*LightGCN: Simplifying and Powering Graph Convolution Network for Recommendation*<sup>1</sup>. We had to read this paper as well and found out that it had a more simple approach to implement than the GTN. The LightGCN model focuses only on the most important component of the GCN which is neighbourhood aggregation for collaborative filtering and pays no attention to the other components feature transformation and nonlinear activation. Since the model is reduced and lighter compared to the other one, our thoughts were that the reproducibility should be easier here. However there were difficulties with this collaborative filtering model as well. The computational power of our devices was insufficient to run the baseline model with the default settings of the algorithm. We decided to stop the process after the program run over the whole night and just finished nearly 15 percent of it. To put it into perspective it took an average of 6 to 7 minutes per epochs meaning around 17 to 18 hours total to get to that point. Furthermore we attempted to adapt the settings to lower standards which was a little bit faster in the process but would not lead to the desired numbers of the paper at all. The screenshots below describing the settings that were used and gives insights of the computational costs that comes with running the model.

## 5 Conclusion

Although the paper was clearly understandable and all of the code and the instructions for the reproduction was given, it was a really difficult challenge to reproduce the results which turned out not to be possible in the end. The reason for that was on one side the high computational effort that came with the proposed approach of the authors. Our computers were not even able to deal with the running process of the baseline models that are already light versions of the algorithm described in the paper. The other reason was lack of documentation and explanation over the provided code. What parameters are there to optimize if we have this capacity problem, should we pass the GPU settings or could be automatically detected.

On the other hand the code provides just the *gowalla* dataset. The code looks not reproducible with

<sup>1</sup><https://arxiv.org/abs/2002.02126>

the other 3 datasets, which were also used in the paper. All in all one can claim that the reproducibility of the paper looks as follows:

1. Reproducible results were **only from gowalla dataset**. (Table 2 from the paper)
2. Reproducible results for effect of the number of layers (Table 3 from the paper) **by changing the parameter  $K$** .
3. Reproducible results for sparsity ratio for gowalla data (Table 4 from the paper) **parameter for changing that ratio was also missing from the input parameters**.
4. Reproducible results for changing the hyper-parameter lambda (Figure 5 from the paper) was possible from the provided input parameters.