# Generalization and Hyper-Parameter Sensitivity of LightGCN

Authors: Daniel Workinn, Erdem Demir, Levent Güner & Xi Chen

## 1 Introduction

Recommending relevant connections in social media services is a challenging problem. By using link prediction techniques, a network's future structure can be predicted and relevant recommendations can be made based on that. On the other hand, classical recommendation systems make predictions based on users' historical behaviors, like most machine learning techniques. The two most popular approaches are content-based and collaborative filtering. Both of the methods have the aim of predicting user preferences for a set of items based on past experiences.

This project will be based on the paper: 'LightGCN: Simplifying and Powering Graph Convolution Network for Recommendation-He et al. [1]'

### 1.1 Aim and Goal

The aim is to run experiments on state-of-the-art Graph Neural Network (GNN) models, using different datasets and hyper-parameter optimization. This will be done to test the method's generalization capabilities as well as its hyper-parameter sensitivity.

The goal of the project is first running these experiments on the LightGCN model using the existing implementation within the Pytorch Geometric library. Then tuning model hyper parameters to evaluate their sensitivity.

### 1.2 Introductory concepts

#### Recommender Systems

A recommendation system is a type of information filtering that attempts to predict how a user will rate or prefer an item, such as a song, service, or movie. Based on a user's previous behavior, a recommendation system makes a prediction. It's intended to predict user preference for a collection of items based on previous experience. Depending on the data used to make inferences, recommender systems can be divided into two categories:

1. Content-based filtering, which uses item attributes,
2. Collaborative filtering, which uses user behavior (interactions) in addition to item attributes

#### Collaborative Filtering (CF)

Collaborative Filtering is a method which creates a user-item matrix in a recommender system. The relationship between users and items is the focus of collaborative-filtering systems. The similarity of items is determined by the similarity of the ratings by the users who have rated the same items. Users are presumed to have similar interests because they share the same interests.

#### Bipartite graph

A bipartite graph is a set of graph vertices that has been dissected into two distinct sets, with no graph vertices in the same set being adjacent. In our case, the bipartite graph structure is intended to be used to represent the user-item graph structure. The user and item embeddings will be propagated over the user-item graph. By doing this, the connections between users and their neighbors will be captured.

### Graph Neural Networks (GNN):

Graph Neural Networks, or shortly GNN are defined as graphs where each node is a recurrent unit, and each edge is a neural network. In each iteration, each node sends a message to its neighbors, and it is propagated through the edges, which are neural networks.

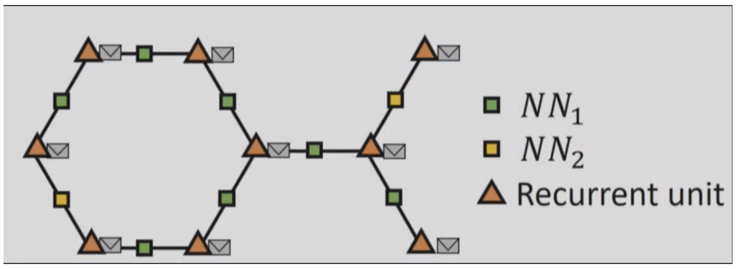

Figure 1: Graph neural networks

### Graph Convolutional Networks (GCN):

Graph convolutional networks is a variant of convolutional neural networks which apply convolution over a graph. It was adopted to solve the problem of node classification [2]. It takes a graph as an input, and for each node, GCN gets the feature information from all its neighbors and itself. Then it averages these values and feeds it into a neural network.

## 2 Review of Relevant Prior Work

### Neural Graph Collaborative Filtering - Wang et al. [3]

This is the main paper that our project is based on. In this paper, integrating the user-item interactions, more specifically, the bipartite graph structure, into the embedding process is proposed. A new recommendation framework, Neural Graph Collaborative Filtering (NGCF) has been developed, which propagates embeddings on the user-item graph structure. Expressive modeling of high-order connectivity in user-item graph is yielded. Experiments on three different public benchmarks are conducted, showing important improvements over several models such as HOP-Rec and Collaborative Memory Network.

### Disentangled Graph Collaborative Filtering - Wang et al. [4]

In this paper, user-item relationships at the finer granularity of user intents were considered. A new model, Disentangled Graph Collaborative Filtering (DGCF), was created to disentangle these factors. A distribution was modeled over intents for each user-item interaction, the intent-aware interaction graphs and representations were iteratively refined. Experiments on three different datasets were conducted for benchmarking, and DGCF achieved significant improvements over several models like NGCF, DisenGCN, and MacridVAE.

### Dynamic Collaborative Filtering - Li et al. [5]

Dynamic Graph Collaborative Filtering (DyGCF) is an unique framework that uses dynamic graphs to describe collaborative and sequential relationships between products and users. In this work, three update mechanisms were proposed: zero-order 'inheritance', first-order 'propagation', and second-order 'aggregation', for the representation of the impact on a user or item based on new interactions. When these interactions occurred in turn, relevant user and object embeddings were updated concurrently, and recommendations were made using the most recent embeddings. The better performance of the model is achieved when the dataset contains fewer actions.

### LightGCN: Simplifying and Powering Graph Convolution Network for Recommendation - He et al. [1]

This paper aims to simplify the design of the Graph Convolution Network (GCN) to make it more concise and appropriate for recommendation. A new model named LightGCN is proposed, including only the “neighborhood aggregation” component in GCN for collaborative filtering. LightGCN is focused on learning user and item embeddings and linearly propagates them on the user-item interaction graph. Therefore, the network uses the weighted sum of the embeddings learned at all layers as the final embedding. A simple model has benefits to implement and train.

# 3 Mathematical Background

## Bayesian Personalized Ranking (BPR) Loss

BPR is a pairwise ranking algorithm that bases on the user’s implicit feedback [8] and aims to provide item recommendations by directly optimizing the rankings. We optimize model parameters with loss function:

$$Loss = \sum_{(u,i,j)\in O} -\ln\sigma\left(\widehat{y_{ui}} - \widehat{y_{uj}}\right) + \lambda\|\Theta\|_2^2$$

where *(u, i, j)* denotes the preference of user *u* to item *i* than *j*, *y* denotes the model predicted estimations of user’s preference, *σ* is the non-linear activation function, and *Θ* stands for all trainable parameters in the model. The gradient of loss is calculated to update the parameter.

## Normalized Discounted Cumulative Gain (NDCG)

NDCG@k is a measure for ranking quality. It measures the performance based on the graded relevance of the recommended top-k items. We can evaluate the effectiveness of the recommender system by calculating:

$$NDCG@k = \frac{DCG@k}{IDCG@k} = \frac{\sum_{i=1}^{p}\left(2^{G(i)} - 1 \big/ \log_2(i+1)\right)}{\sum_{i=1}^{p}\left(2^{|G|(i)} - 1 \big/ \log_2(i+1)\right)}$$

where *G(i)* denotes the grading of item *i*, and |*G*| is the ideal grading set where preference of items is in a descending order.

## Recall

Recall@k is another measure for ranking quality in the top-k recommender system. It is the fraction of relevant items that are retrieved. Getting k nearest neighbors for each sample in the dataset. A neighbor will score 1 if it is matched, otherwise it scores 0. If there is no relevant item in the dataset, recall at k will be set to 1 regardless of the choice of k.

# 4 Algorithm Description

## 4.1 LightGCN

LightGCN aims to simplify the design of GCN by only including its most essential component - neighborhood aggregation - for CF. As described in [1], “LightGCN learns user and item embeddings by linearly propagating them on the user-item interaction graph, and uses the weighted sum of the embeddings learned at all layers as the final embedding”. This linear model is much simpler to implement and train compared to other state-of-the-art GCN recommender models such as NGCF.

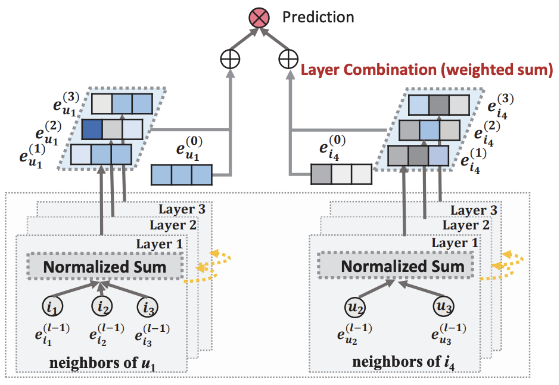

Figure 2: An illustration of the LightGCN model

## 4.2 Main hyper-parameters of LightGCN

**Embedding size**
LightGCN is known to learn user and item embeddings by linear propagation of them on the user-item interaction graph, the weighted sum of the embeddings is being used to learn at all layers as the final embedding [1]. The term "embedding" refers to the low-dimensional, learnt continuous vector representations of discrete variables that are used in neural networks. The dimensionality of categorical variables can be reduced using neural network embeddings, and categories can be properly represented in the transformed space using these embeddings. This makes neural network embeddings a valuable tool [9].

**Batch size**
Batch size determines the number of samples that will be propagated to the neural network in one iteration. As the batch size increases, we have observed a slight decrease in the model's performance, however, the training process was performed faster. This is because of the amount of the data processed at the same time.

**Learning rate**
The learning rate governs how much the model changes each time the model weights are changed in response to the predicted error. A number too small may result in a protracted training process that becomes stuck, while a value too big may result in learning a sub-optimal set of weights too quickly or an unstable training process.

**Weight decay**
Weight decay is a regularization method that adds a penalty term to the cost function. L2 penalty is the most common one and we have used L2 penalty during the experiments. The L2 penalty is represented by

$$C = E + \frac{\lambda}{2n} \sum_{i=1}^{n} w_i^2$$

where $w_i$ are the weights and λ is a manually picked parameter. The greater λ value was reached the better performance, as it is converging faster.

**Layers**
Layer size is determining the depth of the neural network. Since the deeper network has the more parameters, it takes longer time for deeper networks to reach better performances. Therefore, we have observed that the less number of layers reached the better performance faster.

**Epoch**
An epoch is training the network with all the data for a cycle. We have used 100 epochs for all the experiments. However, it is expected to have slightly better performance with greater number of epochs. Due to computational power limitations, it was not possible to perform with greater epochs.

# 5 Dataset Description and Processing

To evaluate the generalization capabilities and hyper-parameter sensitivity of LightGCN, we will conduct experiments on one public dataset - Gowalla. The statistical properties of the dataset are summarized in Table 1.

**Gowalla:** This dataset was collected through the location-based social networking website Gowalla during a 20 month period between 2009 and 2010 [6]. Each user in the dataset has checked into at least one location.

## Processing of the Datasets

The raw dataset contains more information than needed for this project. For this reason, all other data than UserID, ItemID and Timestamp were stripped from the datasets. The data was then converted to the same data-type across the datasets to create a common standard.

Next, the datasets were then sorted by timestamp in ascending order. The sorting was done to maintain the temporal order of the data, which will create a more realistic scenario when deploying the model - train on earlier data, test on later data. Furthermore, the temporal order of the data might also carry some signal useful for the link prediction task.

To ensure the quality of the datasets, 10-core setting [7] was used on both the User and Item nodes. This was done to ensure the quality of the dataset and to use the same settings as the authors in [1], where the latter secure commensurability between the obtained results.

Table1: Statistics of the dataset pre- and post-processing

| **Dataset** | **#Users** | **#Items** | **#Interactions** | **Density** |
|---|---|---|---|---|
| Gowalla RAW | 107,092 | 1,280,969 | 6,442,892 | $4.7 \times 10^{-5}$ |
| Gowalla 10-core | 29,858 | 40,981 | 1,027,370 | $8.4 \times 10^{-4}$ |

# 6 Experiments

In these experiments we want to examine LightGCN's hyper-parameter sensitivity as well as test its generalization capabilities. The settings for the experiments are described in Section 6.1. We then present the results of the experiments in Section 6.2.

## 6.1 Experimental Settings

From the paper [1], we identified the main hyper-parameters of LightGCN to be ***Embedding size***, ***Batch size***, ***Learning rate***, ***Weight decay*** and ***Propagation layers***. These are the five hyper-parameters we are manipulating and recording the performance difference in our experiments. We ran each setting of LightGCN for 100 epochs, which is lower to the proposed 1000 by the authors of [1].

To lower the experimental workload, we decided to not run a full grid search, but rather using the settings proposed in [1] as a starting point and then changing one hyper-parameter at a time. Each run of the LightGCN algorithm on the Gowalla dataset took us about 120 minutes (even when only running 100 epochs instead of 1000) and the full grid search of the five hyper-parameters would have amounted to 243 runs, which was unfeasible. By only changing one hyper-parameter at a time, we reduced the number of runs to 15.

The settings proposed in [1], which we used as a starting point are described in Table 2 below.

Table 2: Hyper-parameter settings proposed by the authors of [1]

| **Embedding size** | **Batch size** | **Learning rate** | **Weight decay** | **Propagation layers** |
|---|---|---|---|---|
| 64 | 1024 | 0.001 | $1 \times 10^{-4}$ | 3 |

The settings used for our experiments are summarized in Table 3 below. We changed one hyper-parameter at a time, while keeping the others fixed to the setting proposed by the authors in [1].

Table 3: Hyper-parameter settings we tested

| **Embedding size** | **Batch size** | **Learning rate** | **Weight decay** | **Propagation layers** |
|---|---|---|---|---|
| 32 | 512 | 0.0005 | $1 \times 10^{-5}$ | 2 |
| 64 | 1024 | 0.001 | $1 \times 10^{-4}$ | 3 |
| 128 | 2048 | 0.005 | $1 \times 10^{-3}$ | 4 |

**Training and Testing**
The gowalla dataset was split into a training and testing set by the same method used by the authors of [1]. This resulted in both the training as well as the testing set including all of the User nodes with the two datasets differing in the Item nodes and edges to them. The training set contained about 77% of the edges while the testing set contained about 23% of the edges.

**Performance Metrics**
To measure change in performance of LightGCN when manipulating the hyper-parameters, we computed and recorded training loss over epochs (during training phase) and recall@20 as well as ndcg@20 over epochs (during testing phase). The result of the experiments can be seen in Section 6.2.

## 6.2 Results

### Embedding size

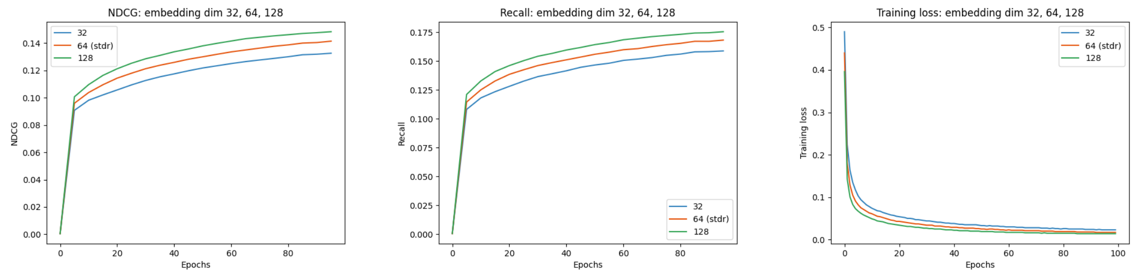

Figure 3: Results of changing embedding size of LightGCN

Figure 3 shows that increasing the embedding size increases the performance of the model. Furthermore, there was no evidence of overfitting during the tests as the test performance does not drop as the number of epochs increases.

### Batch size

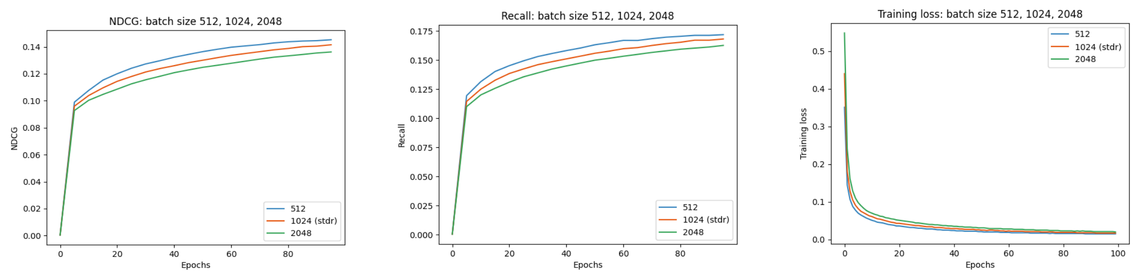

Figure 4: Results of changing batch size of LightGCN

Figure 4 shows that decreasing the batch size increases the performance of the model. Furthermore, there was no evidence of overfitting during the tests as the test performance does not drop as the number of epochs increases.

### Learning rate

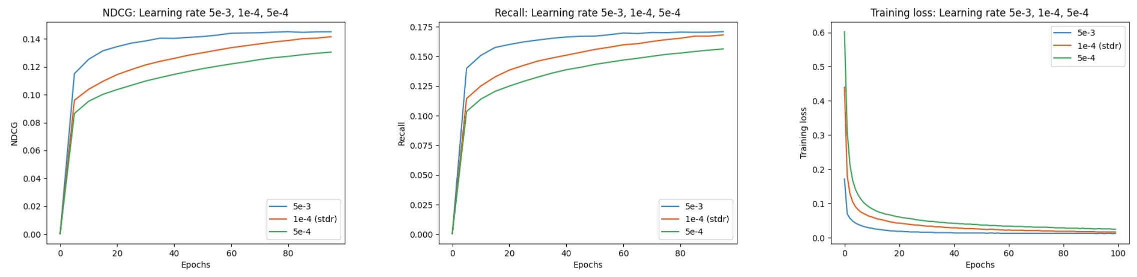

Figure 5: Results of changing learning rate of LightGCN

Figure 5 shows that increasing the learning rate increases the performance of the model. Furthermore, there was no evidence of overfitting during the tests as the test performance does not drop as the number of epochs increases.

## Weight decay

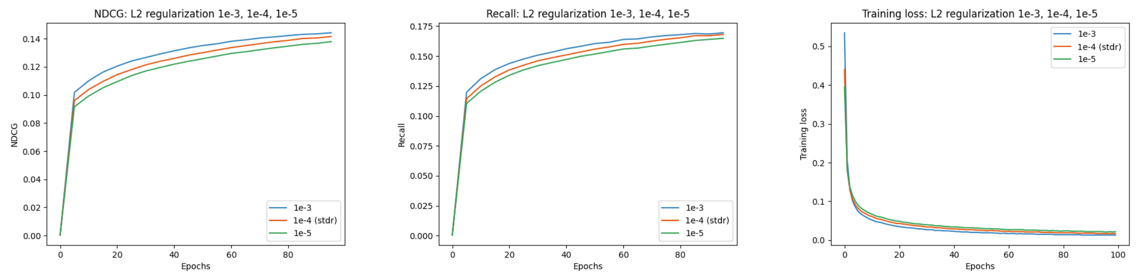

Figure 6: Results of changing weight decay of LightGCN

Figure 6 shows that decreasing the weight decay increases the performance of the model. Furthermore, there was no evidence of overfitting during the tests as the test performance does not drop as the number of epochs increases.

## Propagation layers

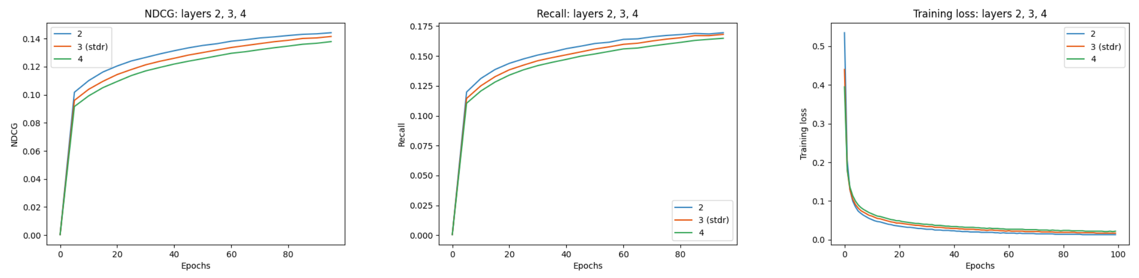

Figure 7: Results of changing number of propagation layers of LightGCN

Figure 7 shows that decreasing the number of propagation layers increases the performance of the model. Furthermore, there was no evidence of overfitting during the tests as the test performance does not drop as the number of epochs increases.

In Table 4 below, the results are summarized in comparison to the hyper-parameter setting proposed by the authors in [1].

Table 4: Results summarized and compared to authors proposed setting

| **Changed Parameter** | **New Setting** | **Recall@20** | **NDCG@20** |
|---|---|---|---|
| Proposed setting | - | 0.1681 (0.0%) | 0.1415 (0.0%) |
| Embedding size | 32 | 0.1587 (0.944%) | 0.1326 (0.938%) |
| Embedding size | 128 | 0.1753 (1.043%) | 0.1484 (1.049%) |
| Batch size | 512 | 0.1718 (1.022%) | 0.1452 (1.026%) |
| Batch size | 2048 | 0.1626 (0.967%) | 0.1361 (0.962%) |
| Learning rate | 0.0005 | 0.1562 (0.930%) | 0.1304 (0.922%) |
| Learning rate | 0.005 | 0.1708 (1.016%) | 0.1450 (1.025%) |
| Weight decay | $1 \times 10^{-5}$ | 0.1684 (1.002%) | 0.1408 (0.995%) |
| Weight decay | $1 \times 10^{-3}$ | 0.1535 (0.914%) | 0.1301 (0.920%) |
| Propagation layers | 2 | 0.1694 (1.008%) | 0.1441 (1.019%) |
| Propagation layers | 4 | 0.1648 (0.980%) | 0.1378 (0.974%) |

## 7 Conclusion and Future Work

The results of our experiments showed that the LightGCN model does not overfit the data given the used hyper-parameter space and running the algorithm for 100 epochs. Therefore, given these experimental settings, the model generalizes well to the testing dataset.

Changing the hyper-parameters from the settings proposed by the authors in [1] showed some difference in the performance metrics recall@20 and NDCG@20. These differences are best summarized in Table 4. The biggest differences were negative ones, particularly when lowering embedding size and learning rate as well as when increasing weight decay. Lowering the embedding size seems to make the mapping between the discrete vector of all locations in the dataset to a continuous 32 dimension vector too limited, making it hard for the model to appropriately represent the locations. Decreasing the learning rate leads to slower convergence of the model. This in combination with our low number of epochs (100), leads to the poor performance. Increasing the weight decay increases the penalty to especially heavy weights of the model, which keeps them smaller. Since the settings proposed by the authors of [1] does not indicate any form of overfitting to the data when running the model for 100 epochs, increasing the weight decay parameter only hinders the performance.

We can also conclude that the hyper-parameter settings proposed by the authors of [1] seem to be good settings, at least for this dataset, given that changing the hyper-parameters did not show any meaningful increases of the performance.

The main difficulty with this project is being able to run the proposed algorithms on the dataset viably. The size of the Gowalla (and similar) dataset(s) lead to big challenges to run the model in a viable timeframe given our limited amount of computational resources. This forced us to limit the search space of hyper-parameters and only focus on one dataset, even though we preferred this project to be more general, covering more datasets and a bigger hyper-parameter space. Future work is needed to paint a clearer picture of the hyper-parameter sensitivity and generalization capabilities of the LightGCN model, including a bigger hyper-parameter space, more datasets and being able to run for many more epochs (1000+).

# 8 Contributions

**Daniel Workinn:**
Review of relevant work, primarily [1] and [3]
Data exploration and preprocessing of data
Built test bench, ran experiments and made plots
Wrote chapter 4 (4.1), 5, 6 and 7

**I.Erdem Demir**:
Wrote introduction (chapter 1) and  Results (chapter 6.2) parts in the final report. Ran experiments with LightGCN using different hyper-parameters

**Xi Chen:**
Reviewed neural graph collaborative filter [3] and graph convolutional network, ran the experiments for hyper-parameter sensitivity evaluation, wrote chapter 3.

**Levent Güner:**
Proposed the project, Review of relevant work [1], [3], [4], [5] as well as wrote chapter 2 and 4 (4.2).

## References

[1] He, X., Deng, K., Wang, X., Li, Y., Zhang, Y., & Wang, M. (2020). Lightgcn: Simplifying and powering graph convolution network for recommendation. ArXiv:2002.02126 [Cs]. http://arxiv.org/abs/2002.02126

[2] Kipf T N, Welling M. Semi-supervised classification with graph convolutional networks[J]. arXiv preprint arXiv:1609.02907, 2016.

[3] Wang X, He X, Wang M, et al. Neural graph collaborative filtering[C]//Proceedings of the 42nd international ACM SIGIR conference on Research and development in Information Retrieval. 2019: 165-174.

[4] Wang, X., Jin, H., Zhang, A., He, X., Xu, T., & Chua, T.-S. (2020). Disentangled graph collaborative filtering. Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval, 1001–1010. https://doi.org/10.1145/3397271.3401137

[5] Li, X., Zhang, M., Wu, S., Liu, Z., Wang, L., & Yu, P. S. (2021). Dynamic Graph Collaborative Filtering. ArXiv:2101.02844 [Cs]. http://arxiv.org/abs/2101.02844

[6] Dawen Liang, Laurent Charlin, James McInerney, and David M. Blei. 2016. Modeling User Exposure in Recommendation. In WWW. 951–961.

[7] Ruining He and Julian McAuley. 2016. VBPR: Visual Bayesian Personalized Ranking from Implicit Feedback. In AAAI. 144–150.

[8] Rendle S, Freudenthaler C, Gantner Z, et al. BPR: Bayesian personalized ranking from implicit feedback[J]. arXiv preprint arXiv:1205.2618, 2012.

[9] Koehrsen, W., 2018. *Neural Network Embeddings Explained*. [online] Medium. Available at: <https://towardsdatascience.com/neural-network-embeddings-explained-4d028e6f0526> [Accessed 10 June 2022].