Reinforced Graph Neural Networks for Collaborative Filtering

ABSTRACT

With the rise in popularity of personalized content, smart recommendations are becoming ever more important to get right; they can drive or slash user interaction. One of the most attractive methods to generate predictive compatibility scores for never-before-seen content is collaborative filtering. In this paper, we introduce a collaborative filtering model based on a graph neural network extended with reinforcements and show that this RGNN model can generate user ratings with state-of-the-art accuracy.[1]

1 INTRODUCTION

Recommendation systems are a core part of many online services today. As more and more content is generated online, providers need a way to decide what content is shown to each of their users to maximize their enjoyment using the service. Recommending the right content can have effect on time spent on the platform as well as improve user experience. The popular video streaming service Netflix even introduced an open competition with a grand prize in 2006 for successfully improving on the accuracy of their collaborative filtering model [2].

In contract to content-based filtering that uses domain knowledge of users and items such as features of the underlying content and tries to match users with item based on this, collaborative filtering (CF) instead uses the history of the user as well as ratings from other users to predict new ratings. This method can therefore have the advantage of not having to extract features from complex media such as videos or items.

In this work we propose reinforced graph neural networks (RGNNs), a novel approach to tackle the collaborative filtering problem. This approach is based on three core mechanisms: (1) a graph neural network (GNN), (2) a simple feed-forward network, and (3) reinforcements. In particular, the GNN utilizes a bipartite graph built from the user-item rating matrix to gather information between neighboring nodes. This information is then used to generate user and item embeddings which are fed directly into the feed-forward network. The resulting prediction from the feed-forward network and the reinforcements are then used in a final linear layer to obtain the predictions of the RGNN.

2 MODELS & METHODS

This section introduces the models and methods used in this work. Firstly, in subsection 2.1 we describe and explore the provided dataset. Secondly, in subsection 2.2 we outline the baseline approaches used, and finally in subsection 2.3 we describe our proposed RGNN architecture.

2.1 Dataset Description

The dataset X used in this work is composed of $|\mathcal{R}| = 1,176,952$ ratings ranging from 1 to 5. The ratings are given on $|\mathcal{I}| = 1000$ different items by $|\mathcal{U}| = 10000$ distinct users giving a sparsity of 11.77% (quotient of total number of entries and observed entries). Compared to the Netflix Prize [2] and MovieLens [8] datasets, we have a significantly denser data matrix which likely affects the choice of suitable algorithms [14].

Before proceeding further, we analyze whether matrix completion is information-theoretically possible. Note that most low-rank matrices have a logarithmic strong incoherence property in the order of $\mu^2 \in O(\log n)$ [5]. Moreover, given the low sparsity of dataset X, we assume that $r = \text{rank}(X) \in O(1)$. By the Candés-Tao theorem [5], we need that the number of observed entries is $|\mathcal{R}| \in \Omega\left(|\mathcal{U}| \cdot \log^7(|\mathcal{U}|)\right)$. Otherwise matrix completion is information-theoretically impossible. Fortunately, according to the theorem the number of observed entries is sufficient in our case to enable matrix completion.

2.2 Baselines

Our novel model is compared against the following baseline approaches used in collaborative filtering.

SVD [12]: Singular value decomposition (SVD) is a technique in which the data matrix (supposed all entries are observed) is decomposed into the product of two orthogonal matrices and a central diagonal one. The Eckart-Young theorem [6] is then used on this decomposition to obtain the best low-dimensional representation of the data. Since the user-item matrix is not complete in our case, the optimization of the recommendation system is done via learning representations of users and items through the alternating least squares (ALS) algorithm or gradient methods. In our implementation SVD uses stochastic gradient descent and is equivalent to probabilistic matrix factorization [19].

SVD++ [13]: In contrast to SVD, SVD++ takes into account a form of implicit ratings. An implicit rating captures the information that some item is rated by some user. The concrete rating is disregarded in this implicit rating. Similarly to SVD, SGD is used to optimize the SVD++ objective.

NMF [16, 22]: Non-negative matrix factorization (NMF) is the problem of approximately factorizing a given matrix by the product of two matrices with non-negative entries. In the context of collaborative filtering, learning such matrices loosely speaking means segmenting the items and users into categories such as genres and preferences. More specifically, the factorization can be interpreted as: (1) How much a user belongs to a certain category, and (2) by which category of users is a certain item preferred.

SlopeOne [15]: This algorithm is arguably the simplest form of item-based collaborative filtering. SlopeOne predicts the non-observed rating of a user-item pair solely based on how another similar item has been rated. The prediction is made via a simple linear regressor of slope equal to 1 (hence, the name) and a bias parameter which is learnt. The simplicity of this model increases computational efficiency. Additionally, overfitting induced by learning the slope vector is reduced.

NCF [9]: As opposed to canonical matrix factorization approaches, neural collaborative filtering (NCF) replaces the traditional sampling from the inner product between latent codes by a multi-layer perceptron (MLP). Contrary to the proposition in the NCF paper, the generalized matrix factorization (GMF) is omitted in

[1]Source code is available at https://github.com/lfwa/reinforced-gnn.

, ,
.

our implementation since the standalone MLP outperformed the combination of the MLP and GMF on our dataset.

GNN [20]: The main idea of a graph neural network (GNN) for collaborative filtering consists of representing users and items as nodes of a bipartite graph. Edges between the two partitions of the graph are weighed by a (normalized) rating given by a user for the item. The GNN architecture progressively aggregates and propagates neighborhood latent information generating embeddings of the users and items. Ratings are predicted using a single linear layer. A more detailed description of the GNN and our implementation is provided in subsubsection 2.3.2.

2.3 Reinforced Graph Neural Networks

We propose the novel architecture: reinforced graph neural networks (RGNNs). The architecture consists of stacking together a graph neural network (as inspired by Wang et al. [20], Wu et al. [21]) with a feed-forward network and adding reinforcements. The graph neural network is the GNN [20] and the feed-forward network is the NCF [9] as described in subsection 2.2. The GNN learns the embeddings of users and items, and these embeddings are then pushed through the feed-forward network of the NCF to make reliable and accurate rating predictions. Additionally, we have derived a similar concept to reliabilities introduced by Bobadilla et al. [4] which we denote reinforcements (see subsubsection 2.3.4). These concepts allow us to construct a novel architecture obtaining the best of both worlds: (1) The strength of graph-extracted embeddings, and (2) the generalization power of deep learning architectures.

2.3.1 Theoretical description. As depicted in Figure 1, the RGNN model consists of four phases. The phases can be described disjointly and are: (1) The mapping phase, (2) the embedding phase, (3) the prediction phase, and (4) the combination phase. Note that the phases (1) - (3), described in algorithm 1, are called GNN + NCF in the remainder of this work. The combination phase (4) predicts a rating from a given output of the GNN + NCF and the corresponding reinforcement. Training the RGNN is done using standard backpropagation minimizing the mean squared error loss.

2.3.2 Mapping and Embedding. The dataset X represents the sparse adjacency matrix of the bipartite graph $G = (U \cup I, E)$, where U denotes the partition of user nodes, I the partition of items, and E the set of edges weighted by the observed user-item ratings (line 1 in algorithm 1). For node $v \in U \cup I$, let $\mathbf{m}^{\ell}$ represent the aggregated neighbor information of node v at layer ℓ, $\mathbf{e}_v^{\ell}$ its hidden state (or embedding) at layer ℓ, and $\mathbf{N}_v$ its neighborhood. During the embedding phase, the GNN + NCF algorithm simulates the aggregation of information in the L-hop neighborhood of node v. This helps extracting higher-order relations between users and items.

The information about nodes in the L-hop neighborhood of some node $v \in U \cup I$ is made iteratively and is shown in lines 4 - 7 in algorithm 1. In particular, at each iteration the aggregated neighbor information $\mathbf{m}_v^{\ell}$ of node v is computed by taking the sum over the neighbor information for each neighbor $i \in \mathbf{N}_v$ normalized by a discount factor of $(|\mathbf{N}_v|\,|\mathbf{N}_i|)^{-\frac{1}{2}}$. The discount factor accounts for the decay of the influence of ℓ-hop neighbors. The neighbor information for a node pair (v, i) is calculated as a weighted combination of the embedding $\mathbf{e}_v^{\ell}$ of node v and the element-wise composition, denoted by $\odot$, of the embedding $\mathbf{e}_v^{\ell}$ of v and the embedding $\mathbf{e}_i^{\ell}$ of the neighbor i. Using the aggregated neighbor information of node v for the ℓ-hop neighborhood and the ℓ-hop embedding, we compute the $(\ell + 1)$-hop embedding. This computation takes the sum of the weighted ℓ-hop embedding and the aggregated neighbor information $\mathbf{m}_v^{\ell}$. Further, the Rectified Linear Unit (ReLU) activation function is applied to obtain the $(\ell + 1)$-hop embedding $\mathbf{e}_v^{\ell+1}$. Note that $\mathbf{W}_1$ and $\mathbf{W}_2$ are the set of parameters that respectively weigh how much the current embedding is influenced by the original features and to what extent the later hops information affects the embedding. An embedding for a node v is then the concatenation of the ℓ-hop embeddings $\mathbf{e}_v^{\ell}$ (line 8 in algorithm 1).

2.3.3 Prediction. For a given user-item pair (u, i), the input to the prediction phase is the embeddings of the user $\mathbf{e}_u^*$ and item $\mathbf{e}_i^*$. The output is then computed by the feed-forward network denoted by **NN** in algorithm 1. In particular, the feed-forward network is the same as the one used for the NCF baseline (see subsection 2.2).

Algorithm 1: GNN + NCF

```
1  Init: Map data matrix $X$ to graph $G = (U \cup I, E)$
2
3  for $v \in U \cup I$ do
4      for $\ell \in [L - 1]$ do
5          $\mathbf{m}_v^{\ell} = \sum_{i \in \mathbf{N}_v} \frac{1}{\sqrt{|\mathbf{N}_v||\mathbf{N}_i|}} \left(\mathbf{W}_1^{\ell}\mathbf{e}_i^{\ell} + \mathbf{W}_2^{\ell}\left(\mathbf{e}_i^{\ell} \odot \mathbf{e}_v^{\ell}\right)\right)$
6          $\mathbf{e}_v^{\ell+1} = \mathrm{ReLU}\left(\mathbf{W}_1^{\ell}\mathbf{e}_v^{\ell} + \mathbf{m}_v^{\ell}\right)$
7      end
8      Concatenate the embeddings: $\mathbf{e}_v^* = \mathbf{e}_v^1 \oplus \cdots \oplus \mathbf{e}_v^L$
9  end
10 Predict Rating: $\hat{y}(u, i) = \mathbf{NN}\left(\mathbf{e}_u^* \oplus \mathbf{e}_i^*\right)$ for user-item pair $(u, i)$
```

2.3.4 Reinforcements. Since the input to the GNN + NCF model simply consists of user-item pairs, we aim at providing additional information to the network. Inspired by the approach of Bobadilla et al. [4], where the authors use the prediction accuracy of a model as an additional input to the final model, we introduce reinforcements. Reinforcements consists of predictions from different baseline models. The reinforcements are then fed into a single linear layer together with the output of the GNN + NCF to yield the final predictions of our RGNN. An example of a reinforcement type is the SVD baseline, where the reinforcements for users $u \in U$ and items $i \in I$ are computed using SVD. Then, when introducing the reinforcement for a given user-item pair in the RGNN, the reinforcement intuitively holds the information on how good the SVD baseline prediction is compared to the prediction of the GNN + NCF. This concept is easily extended to different reinforcement types. In particular, our model supports reinforcements from the SVD, SVD++, NMF, and SlopeOne baselines. Additionally, multiple reinforcements can be used simultaneously. However, as we will see in the experimental evaluation of our model in section 3, the best performance is obtained with a single reinforcement, namely the SlopeOne reinforcement.

2.3.5 Network Architecture. The RGNN uses two different neural network architectures used for the embedding and predictions. This section provides a brief description of both architectures and the layers used.

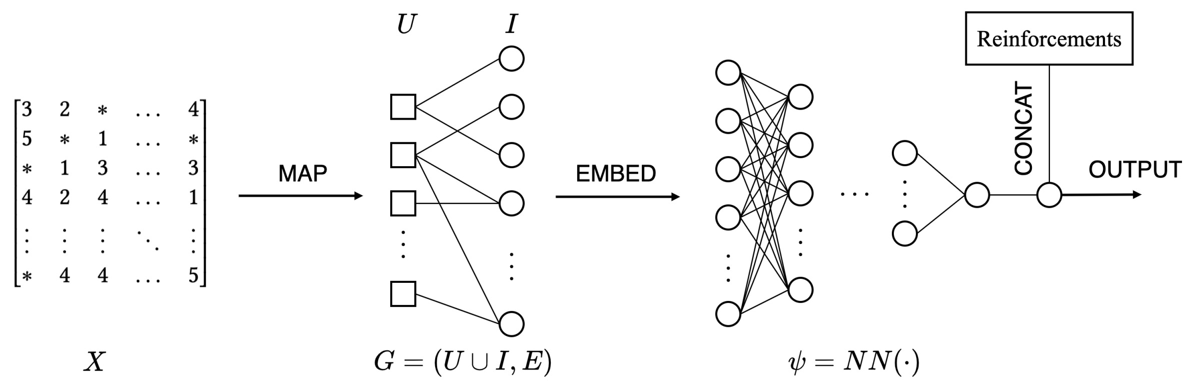

Figure 1: RGNN architecture. (left) Sparse user-item rating data matrix. (center) Bipartite graph representation of the data. (right) Feed-forward neural network that takes the graph-learnt user and item embeddings as well as the reinforcements as input and outputs the rating predictions.

The embedding architecture is the GNN baseline and embeds a user or item using three embedding propagation layers. As mentioned in section 2.3.2, these layers accumulate the information in the three-hop neighborhood of the current node in the bipartite graph. The resulting embedding of a user or item is then of size 3×64.

The prediction architecture is the feed-forward network of the NCF baseline taking as an input the embedding of one user and one item. The network has the following layers: *Dropout, Linear (384), ReLU, Dropout, Linear (64), ReLU, Dropout, Linear (32), ReLU, Linear (16), ReLU, Linear (8), ReLU, Output (1).* The output of the network is then combined using the reinforcements in a final linear layer. Note that as with the NCF architecture, the prediction architecture contains dropout nodes. However, the optimal dropout rate found during hyperparameter tuning was 0.0. This supports the findings of the paper introducing graph neural networks [20].

2.3.6 *Ensemble learning.* We improve predictions by utilizing ensembles of our RGNN model with a simple model averaging method for generating the final predictions. We obtained best results using 12 models with SlopeOne reinforcements and different random seeds. We further tried averaging predictions of RGNNs with different reinforcement combinations, however, this showed inferior performance compared to the SlopeOne reinforcements with different random seeds.

3 EXPERIMENTS

3.1 Experimental Setup

All experiments were performed on the ETH Leonhard cluster[2] using 1 GPU with 64 GB of RAM and 1 CPU. The simple baselines (SVD, SVD++, NMF, SlopeOne) were developed using Surprise [10] and the neural network models (NCF, GNN, GNN + NCF, RGNN) were implemented in the PyTorch Lightning framework [7] for PyTorch [17]. The models were trained for a maximum of 60 epochs using the Adam optimizer [11]. We assume model convergence when no improvement in validation loss was observed for 3 consecutive epochs. The model with the lowest validation loss is used as the final model.

As recommended by Rendle et al. [18], all baselines are extensively tuned to accurately reflect true model performance. Hyperparameter tuning was done using the Optuna framework [1] with a sampler using a Tree-Structured Parzen Estimator [3] to suggest parameters for each trial. No pruning was used to stop unpromising trials early; all trials were completed in their entirety. The optimal hyperparameters can be found in the source code. Similarly, extensive neural architecture searches were performed for the NCF and GNN models.

For the RGNN ensemble we further optimize over the random seeds used for each model by choosing the 12 best scoring seeds. We emphasize that this is only done due to the competitive part of the model scoring and the results do not accurately reflect model performance in different environments.

3.2 Results

We compare the proposed RGNN model to the various baselines outlined in subsection 2.2 as well as the GNN + NCF model. We measure model performance with the root mean square error (RMSE) on the test dataset. Results from our experiments are summarized in Table 1. We observe that combining the GNN and NCF model into the GNN + NCF model yields a significant performance boost compared to the NCF and GNN baselines. Further improvements can be observed when adding reinforcements to the GNN + NCF model leading to our proposed RGNN model. The training time of the RGNN remains similar to the time of the GNN + NCF model. Employing ensemble learning to the RGNN boosts performance from an RMSE of 0.985 to 0.982.

Table 2 shows model performance by reinforcement used in our RGNN model. Adding reinforcements from the SlopeOne baseline yields the lowest RMSE of 0.9854. In Figure 2 we further show a comparison of learning curves between our proposed RGNN model and the GNN + NCF baseline. Here we observe that the loss on the held-out validation set fluctuates less through training when reinforcements are used. Since fluctuations in the validation loss are tied to unstable predictions, we conclude that the RGNN makes more stable predictions than the GNN + NCF.

4 DISCUSSION

From Table 1 we can observe that RGNN performs significantly better than its baseline counterparts. Models combining graph-extracted information with a feed-forward network, i.e. GNN + NCF and RGNN, perform best in our experiments. This suggests

[2]https://scicomp.ethz.ch/wiki/Leonhard

Method	RMSE	Training time (m)
NCF	1.024	65
SVD	1.003	**2**
NMF	1.000	3
GNN	1.000	9
SVD++	0.999	185
SlopeOne	0.998	4
GNN + NCF	0.988	29
RGNN	0.985	26
RGNN ensemble	**0.982**	437

Table 1: Comparison between various baselines and our proposed RGNN method evaluated on the test dataset.

Reinforcements	RMSE
SVD, NMF	0.9880
SVD	0.9875
SVD, SVD++	0.9875
SVD, NMF, SlopeOne	0.9870
SVD, SlopeOne	0.9870
NMF, SlopeOne	0.9870
SlopeOne, SVD++	0.9869
SVD, NMF, SVD++	0.9868
NMF	0.9868
NMF, SlopeOne, SVD++	0.9864
SVD, SlopeOne, SVD++	0.9864
SVD++	0.9862
NMF, SVD++	0.9861
SVD, NMF, SlopeOne, SVD++	0.9859
SlopeOne	**0.9854**

Table 2: Comparison between reinforcements used in our RGNN model evaluated on the test dataset.

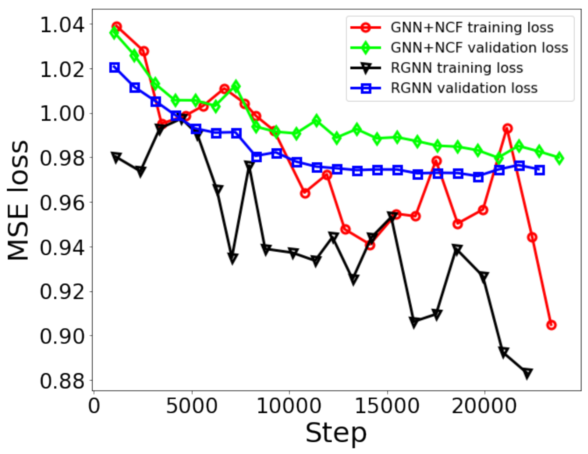

Figure 2: Comparison of learning curves between our RGNN model and the GNN + NCF baseline.

that high-order graph information accurately describes collaborative filtering interactions. Further, we observe that reinforcements improve the performance of the GNN + NCF model. This may be due to the additional information contained in the reinforcements and the final linear layer learning when and how well the user-item pair is representable in the GNN + NCF compared to that of the simple baseline used to generate the reinforcements.

As a word of caution: The previous statement is true only when high quality reinforcements are added to the model. Whenever low quality reinforcements, such as those generated by the SVD baseline, are added, the model may overfit to the reinforcements and perform worse (see Table 2).

Table 1 reports the computation time to run the entire training and prediction process from start to finish. We observe that the neural network models and the SVD++ baseline have the highest time complexity. This may be caused by the neural networks having the largest number of trainable parameters. Conversely, the SVD++ baseline is slow due to not utilizing any fast specialized hardware acceleration such as GPUs. Further, we observe that learning graph embeddings reduces convergence time compared to the standard embeddings in the NCF model.

As already shown by some recent work of Rendle et al. [18], well tuned baselines perform comparably to deep learning models using much less time and resources during the training process. This is also corroborated by our results (see Table 1). The widespread availability of specialized hardware accelerators such as graphics processing units (GPUs) or tensor processing units (TPUs), however, provides wide access to training deep learning architectures with ease like the RGNN proposed in this paper to achieve better performance at the cost of additional training time and resources.

5 SUMMARY AND FUTURE WORK

In this paper, we develop a graph neural network architecture with added reinforcements from simple matrix factorization techniques for collaborative filtering. This method improves upon several baseline approaches by a significant margin on the test dataset (Table 1). Following our ablation study comparing the NCF, GNN, and GNN + NCF baselines, we find that model performance depends on both generating meaningful embeddings of users and items and the expressiveness of the neural network used. We further show that reinforcements generated by the SlopeOne algorithm [15] yields the best results.

In future work, we would like to further explore different types of reinforcements, particularly outputs from more complex models such as neural networks. Similarly, recursive reinforcements can be explored. Extending our experiments to include datasets with a lower sparsity level such as MovieLens can also further investigate the generalizability of our proposed RGNN architecture.

REFERENCES

[1] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. 2019. Optuna: A Next-generation Hyperparameter Optimization Framework. arXiv:1907.10902 [cs.LG]

[2] James Bennett, Stan Lanning, et al. 2007. The netflix prize. In *Proceedings of KDD cup and workshop*, Vol. 2007. New York, NY, USA., 35.

[3] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. 2011. Algorithms for hyper-parameter optimization. *Advances in neural information processing systems* 24 (2011).

[4] Jesus Bobadilla, Santiago Alonso, and Antonio Hernando. 2020. Deep Learning Architecture for Collaborative Filtering Recommender Systems. *Applied Sciences* 10 (04 2020), 2441. https://doi.org/10.3390/app10072441

[5] Emmanuel J. Candes and Terence Tao. 2009. The Power of Convex Relaxation: Near-Optimal Matrix Completion. arXiv:0903.1476 [cs.IT]

[6] Carl Eckart and Gale Young. 1936. The approximation of one matrix by another of lower rank. *Psychometrika* 1, 3 (1936), 211–218.

[7] et al. Falcon, WA. 2019. PyTorch Lightning. *GitHub. Note: https://github.com/PyTorchLightning/pytorch-lightning* 3 (2019).

[8] F Maxwell Harper and Joseph A Konstan. 2015. The movielens datasets: History and context. *Acm transactions on interactive intelligent systems (tiis)* 5, 4 (2015), 1–19.

[9] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural collaborative filtering. In *Proceedings of the 26th international conference on world wide web*. 173–182.

[10] Nicolas Hug. 2020. Surprise: A Python library for recommender systems. *Journal of Open Source Software* 5, 52 (2020), 2174. https://doi.org/10.21105/joss.02174

[11] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[12] Virginia Klema and Alan Laub. 1980. The singular value decomposition: Its computation and some applications. *IEEE Transactions on automatic control* 25, 2 (1980), 164–176.

[13] Yehuda Koren. 2008. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. 426–434.

[14] Joonseok Lee, Mingxuan Sun, and Guy Lebanon. 2012. A comparative study of collaborative filtering algorithms. *arXiv preprint arXiv:1205.3193* (2012).

[15] Daniel Lemire and Anna Maclachlan. 2005. Slope one predictors for online rating-based collaborative filtering. In *Proceedings of the 2005 SIAM International Conference on Data Mining*. SIAM, 471–475.

[16] Xin Luo, Mengchu Zhou, Yunni Xia, and Qingsheng Zhu. 2014. An efficient non-negative matrix-factorization-based approach to collaborative filtering for recommender systems. *IEEE Transactions on Industrial Informatics* 10, 2 (2014), 1273–1284.

[17] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (Eds.). Curran Associates, Inc., 8024–8035. http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[18] Steffen Rendle, Li Zhang, and Yehuda Koren. 2019. On the difficulty of evaluating baselines: A study on recommender systems. *arXiv preprint arXiv:1905.01395* (2019).

[19] Ruslan Salakhutdinov and Andriy Mnih. 2007. Probabilistic Matrix Factorization. In *Proceedings of the 20th International Conference on Neural Information Processing Systems* (Vancouver, British Columbia, Canada) *(NIPS'07)*. Curran Associates Inc., Red Hook, NY, USA, 1257–1264.

[20] Xiang Wang, Xiangnan He, Meng Wang, Fuli Feng, and Tat-Seng Chua. 2019. Neural graph collaborative filtering. In *Proceedings of the 42nd international ACM SIGIR conference on Research and development in Information Retrieval*. 165–174.

[21] Shiwen Wu, Fei Sun, Wentao Zhang, and Bin Cui. 2021. Graph Neural Networks in Recommender Systems: A Survey. arXiv:2011.02260 [cs.IR]

[22] Sheng Zhang, Weihong Wang, James Ford, and Fillia Makedon. 2006. Learning from incomplete ratings using non-negative matrix factorization. In *Proceedings of the 2006 SIAM international conference on data mining*. SIAM, 549–553.