DEEP LEARNING BASED RECOMMENDER SYSTEMS

Tselepidis Nikolaos
ntselepidis@student.ethz.ch

Goetschmann Philippe
pgoetsch@student.ethz.ch

Maksimov Anton
antonma@student.ethz.ch

Pollak Georg Richard
pollakg@student.ethz.ch

Abstract

In this work, we experimentally study four recently proposed classes of deep learning based recommender systems, namely Neural Collaborative Filtering, Collaborative Memory Networks, Neural Graph Collaborative Filtering, and Variational Autoencoders. We focus on collaborative filtering for implicit feedback. Our motivation is based on [3] which states that there is a reproducibility crisis in the field of neural recommendation approaches. Hence, in this work we try to contribute to the resolution of this crisis by objectively evaluating the performance of the selected methods. In order to be able to compare the aforementioned approaches, we modified the authors' implementations so that they can work on the same train-test splits, and also use the same evaluation protocol and metrics. We tuned these models and evaluated their performance on three real world datasets from different application domains. Furthermore, we combined some of the architectures in an ensemble learning context, in an attempt to investigate if this can further boost the recommendation quality. We present extensive comparative results along with discussions.

1. Introduction

Recommender systems are information filtering techniques that aim to predict the level of preference of a user over a specific item. In the era of big data, such techniques have attracted the interest of the scientific community, as they provide a natural approach to improving the user experience on various services, through personalization. Classical recommender systems usually make use of either content-based or collaborative filtering approaches. Content-based filtering techniques utilize specific characteristics of an item in order to recommend additional items with similar properties, while collaborative filtering approaches utilize users' past behaviour i.e. preferences and interactions with items, as well as decisions of other users with similar interests. In most cases, collaborative filtering (CF) techniques yield improved predictions compared to the content-based approaches. There are two main categories of methods when it comes to CF; (i) the Nearest-Neighbor techniques, and (ii) the Matrix Factorization (aka Latent Factor) methods. As the Netflix Prize competition has demonstrated, Matrix Factorization methods are superior to classic Nearest-Neighbor techniques, as they allow the incorporation of additional information to the models, and can thus achieve improved model capacity [8].

Recently, both academia and industry have been in a race to design deep learning based recommender systems in an attempt to overcome the obstacles of conventional models and to achieve higher recommendation quality. In fact, deep learning can effectively capture non-linear and non-trivial user-item relationships, and also enable codification of more complex abstractions as data representations in the higher levels [14]. Various deep neural network architectures have been proposed and shown to be effective for predicting user preferences. Neural Collaborative Filtering (NCF) generalizes the Matrix Factorization (MF) approach by replacing the inner product utilized in MF models by a multi-layer perceptron that can learn non-linear user-item interaction functions, and thus increases the expressiveness of the MF model [6]. Collaborative Memory Networks (CMN) unify the two classes of collaborative filtering models into a hybrid approach, combining the strengths of the global structure of the latent factor model, and the local neighborhood-based structure in a nonlinear fashion, by fusing a memory component and a neural attention mechanism as the neighborhood component [4]. Neural Graph Collaborative Filtering (NGCF) injects the collaborative signal into the embedding process by exploiting the user-item graph structure, so that it can effectively model high-order connectivity in the user-item interaction graph, and thus achieves improved recommendation quality [13]. Other deep learning based recommendation methods include Autoencoders, [12], Variational Autoencoders (VAE), [9], and Restricted Boltzmann Machines (RBMs) [11]. However, as authors have stated in [3] there has been a reproducibility issue with regards to neural recommendation approaches.

In this work, we conduct an objective study of four recently proposed neural recommendation approaches, namely NCF [6], CMN [4], NGCF [13], and VAE [9], that can be used in the context of collaborative filtering for implicit feedback, in an attempt to contribute to the resolution of the reproducibility crisis [3]. It should be stated, that implicit feedback reflects users' preference through behaviours like watching videos, purchasing products, and clicking items [7]. As opposed to explicit feedback, i.e. ratings and reviews, implicit feedback can be tracked automatically and in vast amounts, but is more challenging to utilize, since only user-item interactions are collected instead of user preferences.

In Section 2, we briefly present the aforementioned approaches along with the main underlying concepts. In Section 3, we give extensive comparative results of the selected approaches on three datasets from different application domains, i.e MovieLens (movie recommendations) [5], Epinions (product recommendations) [2], and Jester (joke recommendations) [1]. In Section 4, we discuss the strengths and weaknesses of the selected methods based on the results, and finally, in Section 5, we summarize our work.

2. Models and Methods

1) Neural Collaborative Filtering

The Neural Collaborative Filtering (NCF) approach [6] is tightly related to the Matrix Factorization (MF) method [8], and provides a framework that is able to express MF and also generalize it. Instead of simply "combining" the user and item latent vectors using a fixed inner product, NCF utilizes a multi-layer neural network architecture that learns the interaction function from the data, and thus, increases the expressiveness of the MF model.

An instance of the NCF framework takes as input two binary one-hot encoded vectors, one for the user, and one the item, and passes them through an embedding layer, i.e. a fully connected layer that projects the sparse representations onto dense vectors. The obtained user and item embeddings can be viewed as the user and item latent vectors, respectively. These vectors are then fed into a multi-layer neural network architecture, the output layer of which computes a predicted score $\hat{y}_{ui}$, for the specific user-item interaction. This score represents how likely the item i is relevant to the user u.

Considering the one-class nature of implicit feedback, the value of y_{ui} is being viewed as a label, where 1 means item relevant to u, and 0 otherwise. Therefore, training is performed by minimizing the standard binary cross-entropy loss (a.k.a. negative log loss) between $\hat{y}_{ui}$ and its target value y_{ui}. It should be noted, that in order to treat the problem as a binary classification problem and use the aforementioned loss function, negative instances are also required. These instances are uniformly sampled from the unobserved interactions in each iteration.

In the paper [6], three instantiations of the NCF approach are considered, namely, the Generalized Matrix Factorization (GMF), the Multi-Layer Perceptron (MLP), and the Neural Matrix Factorization (NeuMF). We briefly discuss them below.

GMF: Considering that the output of the embedding layer is the user and item latent vectors p_u and q_i, respectively, we can define the mapping function of the first NCF layer as the element-wise product $p_u \odot q_i$. We can then choose the output layer to be:

$$\hat{y}_{ui} = \alpha_{out}(\boldsymbol{h}^\intercal(\boldsymbol{p}_u \odot \boldsymbol{q}_i)), \tag{2.1}$$

where a_{out} and h denote the activation function and affine transformation weights, respectively. Setting a_{out} to the identity and enforcing h to be a uniform vector of 1, we can exactly recover the MF model. In the GMF model, the authors set a_{out} to the sigmoid function and learn the values of h from the data with the log loss, effectively generalizing the MF approach.

MLP: Instead of computing the element-wise product of the user and item latent vector, in a neural network framework it seems definitely intuitive to concatenate them, and then feed them into a standard MLP to learn the user-item interaction function. In this way, much more flexibility and nonlinearity can be incorporated to the model, compared to the GMF approach, and increased expressiveness can be achieved. In [6], the authors utilize a tower pattern for the layers, halving the layer size for each successive layer. As activation function the use ReLU in the middle layers, and sigmoid in the output layer.

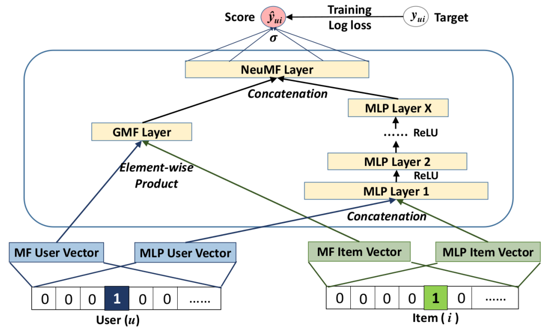


Figure 1: Neural matrix factorization model.

NeuMF: In the NeuMF model, the authors fuse the GMF and MLP approaches into a single architecture (see Fig. 1) in an attempt to combine the linearity of MF and non-linearity of MLP, and thus to be able to better capture complex user-item interactions. They even add more flexibility to the fused model by allowing GMF and MLP to learn separate embeddings. In the last layer, the outputs of GMF and MLP are concatenated and are being fed into a sigle neuron with a sigmoid activation function.

2) Collaborative Memory Network

The Collaborative Memory Network is a CF method and an extensions of NCF [6] that tries to introduce localized user-item interactions using a neighborhood based approach by reweighting with an attention mechanism. It contains three so called "memory" states, a user-, item-and a collective neighborhood memory-state: Let i corresponds to the one-hot encoded item and u to the one-hot encoded user the user and item memory states are then defined by:

$$\boldsymbol{m}_u = \boldsymbol{M}\boldsymbol{u} \qquad \boldsymbol{e}_i = \boldsymbol{E}\boldsymbol{i} \qquad \text{with} \qquad \begin{matrix} \boldsymbol{i} := \mathbb{1}_i & \boldsymbol{M} \in \mathbb{R}^{P\times d} \\ \boldsymbol{u} := \mathbb{1}_u & \boldsymbol{E} \in \mathbb{R}^{Q\times d} \end{matrix}$$

Output Model: the attention mechanism allows to put attention on specific users withing the neighborhood that are similar to the current user of interest. This weighting can be defined by calculating a user-preference vector $\boldsymbol{q}_{ui}$, where each dimension v corresponds to the endorsement of usre v for item i plus the similarity with the target user u:

$$\boldsymbol{q}_{uiv} = \boldsymbol{m}_u^\intercal \boldsymbol{m}_v + \boldsymbol{e}_i^\intercal \boldsymbol{m}_v \qquad \forall v \in N(i)$$

the final weighting is then calculated by taking the softmax: $\boldsymbol{p}_{uiv} = \text{softmax}(\boldsymbol{q}_{uiv})$
This weighting is then used in order to calculate the collective neighborhood memory-state

$$\boldsymbol{o}_{ui} = \sum_{\forall v \in N(i)} \boldsymbol{p}_{uiv}\boldsymbol{c}_v \qquad \boldsymbol{c}_u := \boldsymbol{C}u \qquad \boldsymbol{C} \in \mathbb{R}^{P\times d}$$

The output/rating $\hat{r}_{ui}$ of the model for a given user and item is then given by:

$$\boldsymbol{z}_{ui} = \phi\Big(\overbrace{\boldsymbol{U}(\boldsymbol{m}_u \odot \boldsymbol{e}_i)}^{\text{global use-item interaction}} + \underbrace{\boldsymbol{W}\boldsymbol{o}_{ui}}_{\text{localized user-item-interactions}} + \boldsymbol{b}\Big) \xrightarrow{\text{rating}} \hat{r}_{ui} = \boldsymbol{v}^\intercal \boldsymbol{z}_{ui}$$

$$W, U \in \mathbb{R}^{d\times d}, v, b \in \mathbb{R}^d \text{ are trainable parameters.} \tag{2.2}$$

Multi-Hop Model: multiple of the memory models may be stacked together in order to improve the models predictability:

$$\boldsymbol{z}_{ui}^t = \phi\big(\boldsymbol{W}\boldsymbol{z}_{ui}^{t-1} + \boldsymbol{o}_{ui}^t + \boldsymbol{b}^t\big) \quad \text{with} \quad \boldsymbol{z}_{ui}^0 = \boldsymbol{m}_u + \boldsymbol{e}_i \tag{2.3}$$

3) Neural Graph Collaborative Filtering

Tha main difference of NGCF [13] and NCF is the ""embedding propagation layers"" which are designed to incorporate collaborative signals. This is achieved by including the information about high-order connectivities in users-items connections graph into embeddings of users and items 3. This information isn't used explicitly in other methods, therefore this type of architectue is supposed to improve the quality

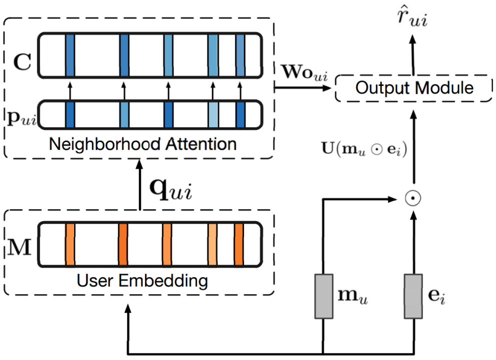


Figure 2: Schematic of the architecture taken from [6]

of recommendations. Specifically, each layer of embbedding propagation layers corresponds to the distance between user and item interaction. It produces messages which are passed between these graph nodes, are summed and then transformed to the embedding at this level. In the end all the

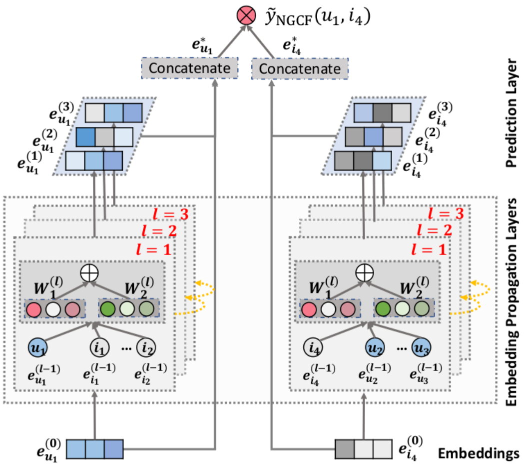


Figure 3: NGCF architecture.

embeddings for each level are concatenated in order to produce a final embedding. This embedding plays the role of latent vectors which can simply be multiplied in order to obtain the rating. The model is trained using pairwise BPR loss and optimized using Adam.

4) Variational Autoencoder

The original authors implement a variational autoencoder to solve collaborative filtering problems that are based on implicit feedback.

Variational autoencoders represent non-linear probabilistic models and can thus capture more complex relationships in the data than the linear factor models which are currently prevalent in collaborative filtering research.

The initialize the item vector by a Gaussian distribution $\sim \mathcal{N}(0, 0)$. This sample is then transformed by a neural network to produce a probability vector over the items in the dataset where the probabilities correspond to the users preferences.

They use a multinomial likelihood to model the user's interaction history and empirically show that it outperforms gaussian and logistic likelihoods.

For inference the parameters of the neural network have to be estimated. For this purpose the posterior distribution of the probability vector over the items given a user and his interaction history needs to be calculated. This is not directly possible. As a solution the authors rely on variational inference which approximates the desired distribution through a fully factorized (diagonal) Gaussian distribution. Variational inference then tries to minimize the Kullback-Leiber divergence between the desired distribution and the surrogate. The new surrogate distribution grows in complexity with the number of users which might become problematic so the authors replace it's parameter by a data-dependent function (referred to as inference model) whose complexity only relies on the number of items in the data. This function introduces a new parameter ϕ.

For learning, the log likelihood of the marginal data is lower bounded through the evidence lower bound which results in a loss function that depends on the parameters of the neural network and ϕ. However, it's not trivially possible to take gradients with respect to ϕ. To solve this issue the authors use the reparametrization trick which makes it possible to take the gradient with respect to ϕ. Now it's possible to train the network with stochastic gradient descent.

Rewrite/split up (way to long and compliate sentence) To make predictions with a trained model, the user's interaction history is needed and put into a certain part of the inference model to calculate the input to the neural network which transforms it into a probability vector that predicts the probabilities with which items the user is most likely going to interact next.

This means, given a new user, only two relatively efficient functions have to be called which makes predictions cheap.

5) Ensemble

For the ensemble method we combined the interaction probabilities of the other methods in the following way: each method generates a vector of interaction probabilities. Those probabilities can also be interpreted as rankings. To combine the rankings we gave weights to each rank. The lowest rank got a weight of zero, the second lowest rank a weight of one, the third lowest rank a weight of two and so forth. To calculate the combined ranks we simply added up the weights given by each method for a particular item. For example, the item with the most added up weight would now have the highest combined rank. Then the combined ranks can now again be interpreted as interaction probabilities.

We choose this method because of it's simplicity and empirically it already delivers interesting results, however there are many ways how rankings or probability vectors can be combined and other methods might yield even better results.

3. Results

1) Experimental Setup

Datasets. We study the effectiveness of the aforementioned neural recommendation approaches on three publicly available datasets, i.e. Movielens [5], Jester [1], and Epinions [2]. The main characteristics of these datasets are summarized in Table 1.

Dataset	#Users	#Items	#Interactions	Density
Movielens	6,040	3,706	1,000,209	0.0447
Jester	24,938	100	616,912	0.2474
Epinions	27,453	37,274	99,321	0.0001

Table 1: Dataset statistics.

Movielens is a movie rating dataset that has been widely utilized as a benchmark for evaluating collaborative filtering algorithms. In our work, we use the version containing nearly one million ratings, where each user has rated at least 20 movies. Jester, on the other hand, is a joke rating dataset with a lot more users, but a lot fewer items compared to Movielens. We use the version where each user has rated between 15 and 35 jokes. Epinions is a dataset containing consumer reviews for various products. This is a very sparse dataset, i.e. most of the users have rated very few items, a fact that leads to the existence of a very weak collaborative signal in the dataset. Therefore, Epinions is a very difficult benchmark for the selected methods, since all of them utilize the collaborative filtering effect, and thus, low quality recommendations are expected.

It should be stated, that although all the aforementioned datasets, include explicit feedback from users, we transformed them into implicit feedback datasets, in order to study the learning from the implicit signal. To this end, we binarized the ratings, i.e. whenever there is a rating of a user to an item, either positive or negative, we set it to 1, since it denotes the existence of a user-item interaction. If there is no such interaction we set it to 0.

Evaluation. We evaluate the quality of item recommendation using the leave-one-out evaluation method, following the prior work [6, 4]. In order to make the split as realistic as possible, for each user we held-out their latest interaction as the testset, and utilized the remaining data for training. Then, we ranked the "positive" test item (i.e. item with the latest interaction by the user) against m randomly sampled "negative" items (i.e. items that this user has never interacted with). We evaluated the ranking quality using the Hit Ratio (HR@k), and the Normalized Discounted Cumulative Gain (NDCG@k) metrics. Intuitively, HR@k measures the presence of the "positive" item within the top k items, while NDCG@k measures the items' position in the ranked list, penalizing the score for ranking the item lower in that list. We computed both metrics for each test user and for k=10, and reported the average score.

It should be stated, that since the Jester dataset does not include timestamps, we generated a random timestamp for each rating, and then proceeded to the train-test split. Moreover, it should be mentioned that for the case of Epinions, we filtered out from the dataset all the users that have only rated a single item, so as to avoid the cold-start setting (for the users). Finally, for Movielens and Epinions we set m=99, while for Jester we set m=49, since there are only 100 items in the dataset in total, while there are users that have rated up to 35 items.

2) Performance Comparison

In Tables 2, 3, and 4, we present the best HR@10 and NDCG@10 scores achieved by each neural recommendation method, including the ensemble learning approach, on the Movielens, Jester, and Epinions datasets, respectively. Fig. 4, depicts the HR@k and NDCG@k scores while varying k from 1 to 10. In both the tables and the plots, along with the methods' names we state the most important parameters that we utilized in order to achieve each score. We refer to the GMF method with embedding size e as GMF(e). Similarly, we do for the NGCF and CMN approaches. We refer to the MLP with l layers as MLP(l). It should be noted, that for the MLP method we follow a tower pattern for the layers as described in Section 2. Moreover, the size of the embeddings is always set to be half the width of the first layer, while the width of the output layer is fixed to 8. For the NeuMF approach with embedding size e in the GMF component, and l layers in the MLP component we write NeuMF(e,l). For all methods that use embedding layers we tried the values e=8,16,32,64 for tuning. For the MLP and NeuMF approaches we varied the number of layers from 2 to 4. For the VAE method we used the standard configurations.

3) Comments

On train-validation-test splits for VAE: In the original implementation of VAE, the authors have done their train-validation-test splits in a different way compared to our split. They splitted the users in respective groups instead of excluding some interactions for each user. For our comparison this way of splitting was not possible because some methods need to train embeddings for their users. As a result, the HR and NDCG scores that were computed with the validation split for VAE, did not agree with the respective scores computed on the test split. The validation metrics became worse as we trained for more epochs, however the test metrics became better. To not let the VAE underperform, we tuned the number of epochs according to the test-split. This also suggests that we are indeed overfitting on the users we are training on (which is also on what the test metrics are evaluated). This might or might not be desired in a real-world application.

Movielens	HR@10	NDCG@10
GMF(32)	0.7023	0.4202
MLP(4)	0.6616	0.3897
NeuMF(32,2)	0.7012	0.4233
CMN(64)	0.6572	0.3882
NGCF(64)	0.6937	0.4122
VAE	0.7001	0.4230
Ensemble	**0.8412**	**0.6288**

Table 2: Best performance achieved by each method on Movielens dataset.

Jester	HR@10	NDCG@10
GMF(16)	0.8411	0.7589
MLP(3)	0.8427	0.7569
NeuMF(8,2)	0.8404	0.7563
CMN	−	−
NGCF(32)	0.8558	0.7615
VAE	0.8482	0.7648
Ensemble	**0.8802**	**0.7686**

Table 3: Best performance achieved by each method on Jester dataset.

−: CMN took 24h for three epochs and did not lead to any useful results

Parameter tuning for VAE: The authors state that annealing is quite important for the performance of VAE, so we tried different largest annealing parameters ranging from 0.0 to 1.0, but neither validation- nor test-metrics were influenced by it.

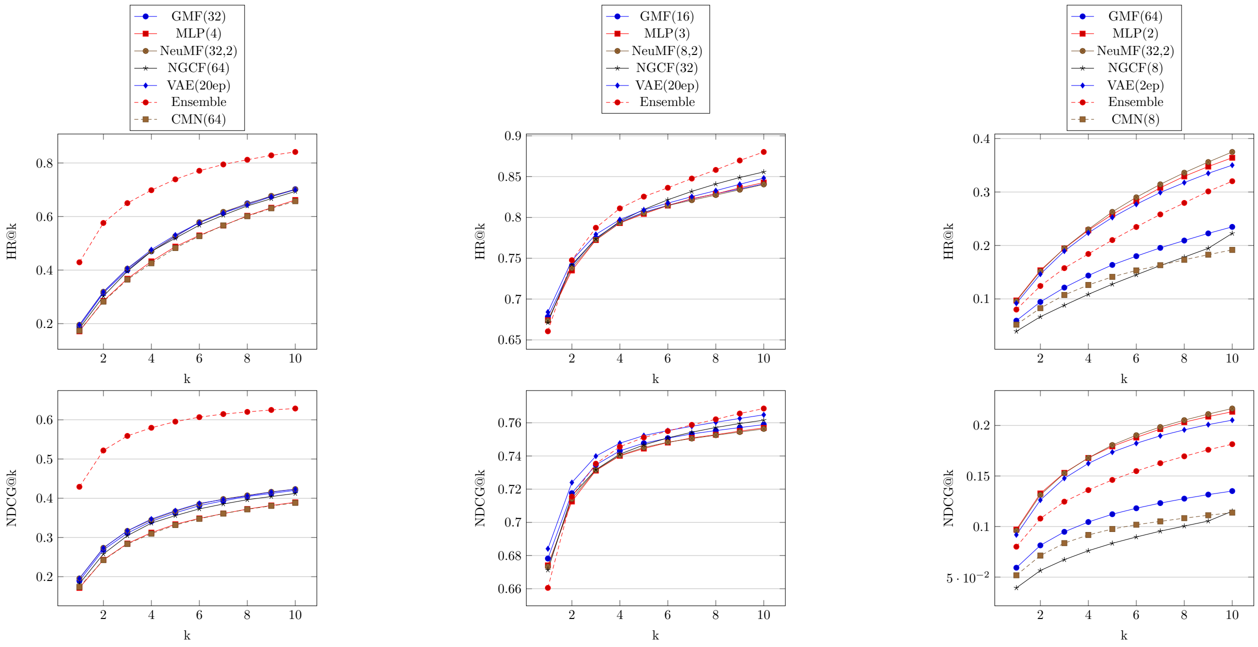
Ensemble: We decided not to include the CMN into the ensemble as it did not improve prediction accuracy.

CMN difficulties: The fact that the method took 24h in order to compute three epochs for the jester data set can most likely be contributed to the density of the data set as well as the computational complexity of the method $\mathcal{O}\left(d|N|(i) + d^2 + d\right) \propto |N(i)|$. It seems also to be logic that the attention mechanism needs a lot of training interactions in order to perform well, which can be seen by the improved accuracy of the movielens data set. One thing that has to be mentioned is that we could not reproduce the same results as the original paper neither on our epinions data split nor on the original data (as the epinions data was missing).

4. Discussion

The NCF methods, namely GMF, MLP, and NeuMF, although relatively simple, seem to yield satisfactory results for all datasets, compared to the other approaches. In fact, for Movielens, the GMF and NeuMF methods lead to quite high HR and NDCG scores, that are only surpassed by the ensemble approach. For Jester, they all yield very good results not very different from the other methods. For Epinions, MLP and NeuMF seem to achieve the highest scores. We need to mention that all of the NCF approaches required nearly 20 to 40 minutes to train, and that the most effective updates occur in the first 10 epochs (similar to what the authors state).

The NGCF approach shows itself among other assessed methods as the best on the Jester dataset and as very good on the Movielens, but as the worst on the Epinions dataset. It could be explained with differences in density of these



(a) Movielens **(b)** Jester **(c)** Epinions

Figure 4: Evaluation of top k item recommendation, where k ranges from 1 to 10 on the three datasets.

Epinions	HR@10	NDCG@10
GMF(64)	0.2348	0.1351
MLP(2)	0.3641	0.2135
NeuMF(32,2)	0.3750	0.2167
CMN(8)	0.1917	0.1138
NGCF(8)	0.2226	0.1152
VAE	0.3503	0.2052
Ensemble	0.3202	0.1814

Table 4: Best performance achieved by each method on Epinions dataset.

datasets, because Epinions is very sparse and, probably, it's hard for NGCF to incorporate collaborative signal into embeddings in these circumstances. Also, training takes from about an hour (Jester and Epinions) to 6 hours (Movielens), what is, in general, worse than other methods.

The VAE performed pretty well compared to the other methods on all tested datasets. Only on Epinions it falls a bit short. One of it's main strength is certainly that even on a i7-3615QM quad-core CPU with 8GB of RAM it could be trained and evaluated in minutes on Movielens and Jester, and in about an hour on Epinions.

Combining different CF methods experimentally seems to work since the ensemble approach managed to achieve the best scores on the Movielens as well as Jester datasets. A little experimenting on the subsets of the methods showed that the increased scores mainly come from combining the VAE with one of the other methods. This suggests that the VAE indeed finds different interesting relationships in the data than the other methods. However, combining methods at such a high level like we did also bears problems, mainly the performance and flexibility will be dictated by the slowest and least flexible method in the ensemble. The decrease (compared to the individual methods) on the Epinions dataset might be related to the fact that our individual methods already perform pretty poorly. Maybe a certain performance threshold on the individual methods is necessary for the ensemble method to have increased performance.

5. Summary

TODO: *Here we will summarize our work. We conducted an objective experimental study of the methods in order to contribute to the resolution of the reproducibility crisis. What were our main conclusions? Furthermore, in order to be able to objectively compare the methods some modification/standarization of the authors' codes were needed. We need to briefly discuss these things.*

Appendix

References

[1] Anonymous Ratings Data from the Jester Online Joke Recommender System. https://goldberg.berkeley.edu/jester-data/. Accessed: 2019-12-12.

[2] Recommender Systems Datasets. http://cseweb.ucsd.edu/ jmcauley/datasets.html#social_data. Accessed: 2020-01-12.

[3] Maurizio Ferrari Dacrema, Paolo Cremonesi, and Dietmar Jannach. Are we really making much progress? A worrying analysis of recent neural recommendation approaches. In *Proceedings of the 13th ACM Conference on Recommender Systems*, pages 101–109. ACM, 2019.

[4] Travis Ebesu, Bin Shen, and Yi Fang. Collaborative Memory Network for Recommendation Systems. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, pages 515–524. ACM, 2018.

[5] F Maxwell Harper and Joseph A Konstan. The Movielens Datasets: History and Context. *ACM Transactions on Interactive Intelligent Systems (TIIS)*, 5(4):19, 2016.

[6] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. Neural Collaborative Filtering. In *Proceedings of the 26th international conference on world wide web*, pages 173–182. International World Wide Web Conferences Steering Committee, 2017.

[7] Yifan Hu, Yehuda Koren, and Chris Volinsky. Collaborative filtering for implicit feedback datasets. In *2008 Eighth IEEE International Conference on Data Mining*, pages 263–272. IEEE, 2008.

[8] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix Factorization Techniques for Recommender Systems. *Computer*, (8):30–37, 2009.

[9] Dawen Liang, Rahul G Krishnan, Matthew D Hoffman, and Tony Jebara. Variational Autoencoders for Collaborative Filtering. In *Proceedings of the 2018 World Wide Web Conference*, pages 689–698. International World Wide Web Conferences Steering Committee, 2018.

[10] Andriy Mnih and Russ R Salakhutdinov. Probabilistic Matrix Factorization. In *Advances in neural information processing systems*, pages 1257–1264, 2008.

[11] Ruslan Salakhutdinov, Andriy Mnih, and Geoffrey Hinton. Restricted Boltzmann Machines for Collaborative Filtering. In *Proceedings of the 24th international conference on Machine learning*, pages 791–798. ACM, 2007.

[12] Suvash Sedhain, Aditya Krishna Menon, Scott Sanner, and Lexing Xie. AutoRec: Autoencoders Meet Collaborative Filtering. In *Proceedings of the 24th International Conference on World Wide Web*, pages 111–112. ACM, 2015.

[13] Xiang Wang, Xiangnan He, Meng Wang, Fuli Feng, and Tat-Seng Chua. Neural Graph Collaborative Filtering. *arXiv preprint arXiv:1905.08108*, 2019.

[14] Shuai Zhang, Lina Yao, Aixin Sun, and Yi Tay. Deep Learning based Recommender System: A Survey and New Perspectives. *ACM Computing Surveys (CSUR)*, 52(1):5, 2019.