Does Deep Learning really improve standard Collaborative Filtering strategies? An analysis on explicit feedback

Francesco Lauria
STUDENT NUMBER: 2041074

THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE IN COGNITIVE SCIENCE & ARTIFICIAL INTELLIGENCE
DEPARTMENT OF COGNITIVE SCIENCE & ARTIFICIAL INTELLIGENCE
SCHOOL OF HUMANITIES AND DIGITAL SCIENCES
TILBURG UNIVERSITY

Thesis committee:

Cicek Guven, Ph.D.
Yash Satsangi, Ph.D.

Tilburg University
School of Humanities and Digital Sciences
Department of Cognitive Science & Artificial Intelligence
Tilburg, The Netherlands
May 2020

Preface

The author would like to thank Cicek Guven, Ph.D, for her precious feedback and encouragement throughout the realization of this project. The author also thanks his family for the support he received during the entire academic year and finally, the author gives a special thanks to his wife for being always supportive and for accompanying him until the realization of this study.

Does Deep Learning really improve standard Collaborative Filtering strategies? An analysis on explicit feedback

Francesco Lauria

Given the positive results of applying Deep Learning (DL) to fields like Computer Vision and Speech Recognition, in recent years, the Recommender Systems (RSs) community has decided to apply DL components to improve the overall performance of RSs models. Among the most recent supporters of DL techniques there are Howard and Gugger (2020) and Ahamed and Afroge (2019), who applied He et al. (2017)'s Neural Collaborative Filtering (NCF) framework to explicit feedback. These studies support the idea that applying neural networks to standard Matrix Factorization (MF) techniques yields better results than using standard techniques. Inspired by Dacrema, Cremonesi, and Jannach (2019), who proved that standard techniques can beat DL-enhanced models, if an accurate experiment is conducted, this work asks the following question: "can a properly tuned standard MF technique outperform recent DL-enhanced RSs based on explicit feedback?" This study tries to overcome some of the shortcomings of Howard and Gugger (2020) and Ahamed and Afroge (2019)'s models, by using a more challenging dataset (the Netflix dataset), a strong baseline for the given dataset (ImprovedRSVD) and by tuning the models with different configurations of hyper-parameters in two consecutive experiments. The results showed that the baseline reached a test RMSE of 0.98 while the best DL model reached a test RMSE of 1.00, therefore proving that a simple standard technique can beat DL enhanced models by performing consistently better than them, even after two successive tuning stages.

1. Introduction

Recommender Systems (RSs) have come to be a key element of many businesses' operations since the mid-1990s (Resnick et al. 1994), (Hill et al. 1995), (Shardanand and Maes 1995). For example, Neil Hunt (Chief Product Officer Netflix) pointed out that 80% of movies watched on Netflix comes from its RS (Smith and Linden 2017). Moreover, RSs are an increasingly prominent part of the web, accounting for up to a third of all traffic on several of the world's most popular sites (Sharma, Hofman, and Watts 2015). In a continuous effort to improve RSs performance, companies such as Amazon, eBay and Netflix have included recommendation techniques into their RSs "to estimate the potential preferences of customers and recommend relevant products or items to the user. Recommendation performances have huge impact on the commercial success of these companies in terms of revenue and user satisfactory" (Wei et al. 2017). In addition to that, RSs are widely used in critical domains like education and government (Patel, Desai, and Panchal 2017). Particularly in light of the more recent application of Deep Learning (DL) to RSs, it has become important to understand which models allow for higher accuracy, in order to support companies' and governments' investment decisions. The development of DL techniques has led many authors to support the idea that DL is indeed the most advanced technique to build RSs (Zhang et al. 2019). This study aims at testing some of the most recent RSs based on DL against standard methods to give some insights into the effectiveness of DL techniques over standard algorithms.

Standard techniques to build RSs can be divided into 3 categories: Content-based filtering (CBF), in which each item is labelled according to its particular characteristics or features (when a user likes, saves or buys an item, the RS will suggest items with similar characteristics); Collaborative filtering (CF), in which the utility of items for a particular user is predicted based on the items previously rated, liked or bought by other users; and Hybrid filtering (HF) techniques, which draw from the strengths of both CBF and CF models. There are two approaches to developing a CF strategy: a) memory-based (data are used directly to predict the recommendation); b) model-based (the data are used to feed another model, which will then predict the recommendation) (Adomavicius and Tuzhilin 2005).

Collaborative Filtering (CF) techniques have become the dominant framework for RSs, most likely because of *de facto* predominance of the user-item (U-I) matrix, to encode the individual preferences of users for items in a collection (Shi, Larson, and Hanjalic 2014). Within CF strategies, Matrix Factorization (MF) has been recognized as one of the most popular approaches to developing a model-based RS. The main benefit of this strategy is that it is efficient in handling high dimensional data (Mehta and Rana 2017). MF was popularized by the Netflix Prize competition in 2006 (Netflix 2006), and has since become the reference point for further experiments to increase RSs performance.

Given the positive results of applying DL to Computer Vision, Natural Language Processing and Speech Recognition (Collobert and Weston 2008), (He et al. 2016a), (Hong et al. 2015), in more recent years, the RS community decided to apply DL components to try to improve the overall performance of RSs models. In particular, He et al. (2017) stood out, as they "strive[d] to develop techniques based on neural networks to tackle the key problem in recommendation — collaborative filtering — on the basis of implicit feedback". They proposed a framework called Neural Collaborative Filtering (NCF), which set the foundation for a series of further contributions (among others Xue et al. (2017),Lian et al. (2017), Wang et al. (2017), Zhang et al. (2018)). These authors thus supported the idea that DL significantly enhanced the standard CF strategies. However, not all authors seemed to agree with He et al. (2017). Specifically, Dacrema, Cremonesi, and Jannach (2019) showed that recent RSs based on DL are not always better than standard techniques, for instance, they proved that the NCF framework developed by He et al. (2017) can be beaten by a simple linear model, called SLIM, which had been developed by Ning and Karypis (2011). A more detailed explanation of these two fundamental papers can be found in the Related Work section.

The present work relies on movie ratings from the Netflix dataset, on the basis of explicit feedback. Authors who modelled explicit feedback and expanded on the work by He et al. (2017), are Ahamed and Afroge (2019) and Howard and Gugger (2020). Ahamed and Afroge (2019) started from the NFC framework to develop the Neural Matrix Factorization (NMF) model for explicit feedback; while Howard and Gugger (2020) developed a DL API based on PyTorch, which includes two specific functions to develop RSs based on DL, namely *EmbeddingDotBias* and *EmbeddingNN*, which will be referred to from now on as MF Fastai and MLP Fastai models. These models will be explained in more details in section 3.2.

Starting from the methodological viewpoint of Dacrema, Cremonesi, and Jannach (2019), who had criticized the results achieved by He et al. (2017), the current work identifies some of Ahamed and Afroge (2019) and Howard and Gugger (2020)'s shortcomings: the use of a weak baseline, the lack of a validation set to tune the models, the reliance on a simple dataset. These limitations will be discussed in the Related Work section. By identifying these limitations and offering alternative ways to test the algorithms, this research aims at giving a contribution towards the debate regarding the alleged superior effectiveness of DL-enhanced CF with explicit feedback over standard techniques. Specifically, the research question is as follows:

> **Can a properly tuned standard MF technique outperform recent DL-enhanced RSs based on explicit feedback?**

The research question has been answered by carrying out two experiments. The first experiment compares the performances of Howard and Gugger (2020)'s models and Ahamed and Afroge (2019)'s NMF model with ImprovedRSVD's performance, which is a well-known and robust baseline for the Netflix dataset (Paterek 2007). All three DL models have been tested in the first experiment with different configurations. The second experiment carries out a further test on the DL models that make use of a MLP extension specifically (MLP Fastai and NMF), to achieve a decisive indicator that standard techniques like ImprovedRSVD stand a chance against DL-enhanced models. These two models were tested, first with a different number of hidden layers, and then with a different number of latent factors. The code used to carry out these two experiments has been uploaded on GitHub and it is available for consultation at Lauria (2020). The results of the first experiment show that the baseline, ImprovedRSVD, proved to be the top performer, as it achieved the lowest RMSE value. This result provides sufficient evidence to answer the research question and state that DL-enhanced CF models do not necessarily perform better than simpler model-based RSs, if a more rigorous procedure is applied, as opposed to Howard and Gugger (2020) and Ahamed and Afroge (2019)'s procedure. This finding is significant to the general discussion related to the alleged superiority of DL based RSs over standard methods, while also supporting and expanding on the findings of Dacrema, Cremonesi, and Jannach (2019). The second experiment reinforces the findings of the first experiment: despite the re-tuning phase, in which different numbers of latent factors and hidden layers were tested, the DL algorithms did not beat the baseline. Finally, space for improvement and for future research is considered in the Discussion section and in the Conclusion.

2. Related Work

Collaborative Filtering (CF) techniques have become the dominant framework for RSs, and within CF strategies, Matrix Factorization (MF) has been recognized as one of the most popular approaches to developing a model-based RS. Among standard techniques to implement MF, Singular Value Decomposition (SVD) received a lot of attention (Ranjan et al. 2019). The first time that SVD was used to solve a recommendation problem was in Sarwar et al. (2000) and it later became popular thanks to the version proposed by Funk (2006) in the Netflix Prize competition. Funk improved Sarwar et al. (2000)' version, which suffered of overfitting and distorted data due to imputation of missing values before applying SVD, therefore opening up the way to attempts by other researchers to improve the SVD method even more (Paterek 2007), (Koren 2008).

More recent attempts to implement MF have leveraged deep learning to enhance CF capabilities (Singhal, Sinha, and Pant 2017). For instance, notable publications took inspiration from techniques such as word2vec to embed users, items or context profiles in latent spaces (Vasile, Smirnova, and Conneau 2016), (Vuurens, Larson, and de Vries 2016). According to Karatzoglou and Hidasi (2017), word2vec embeddings are somewhat more flexible then the standard MF models often employed in CF. These embeddings can be either used directly to predict recommendations or they can be used as input to other DL methods. For example, word2vec embeddings have been used as input to Feedforward Networks, a type of deep learning model that helps discover non-linearity interactions, and therefore makes it possible to understand complex user/item interactions (Zhang et al. 2019). These methods are believed to outperform standard model-based CF methods and ultimately achieve better prediction performance (Karatzoglou and Hidasi 2017).

The current state of the art of DL applied to CF is believed to be the framework developed by He et al. (2017). He et al. (2017)'s work stands out thanks to their Neural Collaborative Filtering (NCF), a specific CF framework based on DL, presented at top-level research conferences. It opened the way for following research and constitutes the foundation of the academic debate at the heart of the current study. In their work, DL was applied for the first time directly to the user-item matrix interaction rather than on auxiliary data like in other studies (e.g. (Cao, Yang, and Liu 2017) and (Li, Kawale, and Fu 2015) among others). He et al. (2017) tackled the problem of recommendation by considering implicit feedback: a value of 1 is assigned if the user has interacted with the item, while a value of 0 is assigned if the user has not interacted with the item. The authors proposed three models:

1. **Generalized Matrix Factorization (GMF):** it replicates MF by element-wise product of the user-item vectors, which are found by embedding layers, and a sigmoid function is used as output layer in order to introduce non-linearity.
2. **Multi-Layer Perceptron (MLP):** the authors add hidden layers on the concatenated vector, which comes from GMF, to learn the interaction between user and item latent features. They further use ReLU as activation function for the hidden layers, and sigmoid for the output layer.
3. **Neural Matrix Factorization (NeuMF):** it is essentially a fusion of GMF and MLP. They are trained with separate user and item embeddings and eventually their outputs are concatenated in the final NeuMF layer, which makes use of a sigmoid as activation function. Fig.(1) summarizes the NeuMF architecture.

He et al. (2017) tested all three proposed models on the MovieLens 1M dataset against two standard MF techniques used for implicit feedback: BPR (Rendle et al.

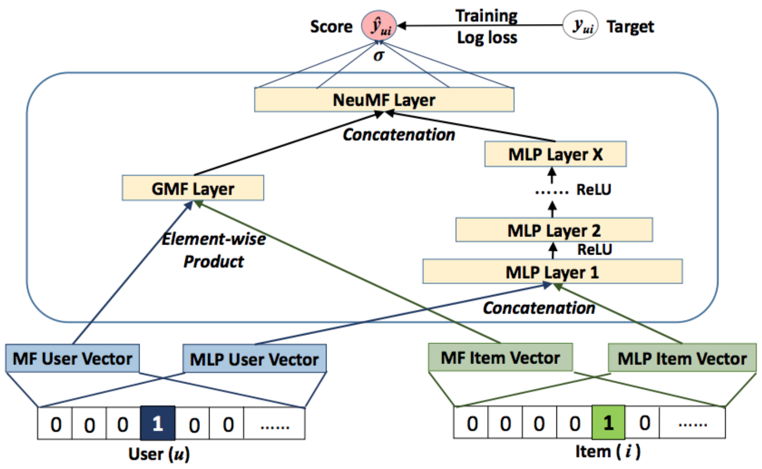


Figure 1: NCF Framework (He et al. 2017).

2012) and eALS (He et al. 2016b), among others baselines. NeuMF achieved the best performance, significantly outperforming eALS and BPR by a large margin. Also, GMF and MLP, The other two NCF methods, performed very well, while between them GMF slightly outperform MLP. However, the authors pointed out that this result could change, because MLP can be further improved by adding more hidden layers: they tested up to 4 hidden layers with increasing numbers of units per layer, showing that "stacking more layers [is] beneficial to performance" ((He et al. 2017)). Generally speaking, their work helped to popularize the NCF framework, and therefore DL techniques, as superior to standard MF algorithms.

As mentioned before, He et al. (2017) opened the way for future research, as many papers agreed with their assumptions and findings. Ahamed and Afroge (2019) proposed a similar algorithm, but they used it to model explicit data, that is, when users rate items assigning specific scores. They introduced one new model called Neural Matrix Factorization (NMF), which uses the NCF framework. The MF layer replicates a basic MF technique but it is implemented with embedding layers and, unlike the MF developed for the NCF framework, non-linear functions were not added. The MLP layer is essentially equal to the MLP of the NCF framework but ReLU is used as output layers instead of sigmoid. Finally, the NMF layer takes the concatenation between the MF layer output and MLP layer output as input and it passes this concatenation to a dense layer before predicting the final rating with a ReLU as activation function. Basically, the main differences between NMF and NCF are:

- the type of network input, which is implicit in the case of NCF and explicit in the case of NMF,
- the type of activation function for the output layer, which is sigmoid for NeuMF and ReLU for NMF.

In this case the three models (MF, MLP and NMF) were compared to standard memory-based models and proved to be better on the MovieLens 100k dataset.

Yet another example of research that followed in the footsteps of He et al. (2017), and creates models for explicit feedback, is Fastai, a DL API based on PyTorch (Howard and Gugger 2020). Among the many functions developed in Fastai, there are two specific functions that use DL to create CF RSs, dealing with explicit feedback. The first is based on embedding layers for users, items and biases plus an output layer implemented with sigmoid; the other one is based on MLP like the MLP model of Ahamed and Afroge (2019), but with sigmoid function as output layer. The former was tested for educational purposes on GitHub (Howard and S.Gugger 2019), using the MovieLens 100k and it showed better results than standard benchmarks. The latter function was never tested. In both cases, there is no evidence of academic research on the topic being published in a journal.

However, not all authors agree on the effectiveness of using DL components to develop RSs. Dacrema, Cremonesi, and Jannach (2019) analyzed 18 DL algorithms for top-n recommendation tasks that were presented at top-level research conferences over the past years. They pointed out that most of the reviewed works can be outperformed, at least on some datasets, by conceptually and computationally simpler algorithms. Particularly, the authors attribute this phenomenon to three different factors. The first is the weakness of baselines. Sometimes the chosen baselines are too weak for the given task and therefore, even if the DL model performs better than the proposed baselines, there could be a non-neural stronger baseline that behaves better than the DL enhanced RS. The second factor is the establishment of weak methods as new baselines. Often authors do not use a validation set to tune the baselines, which means that there could be a better set of hyper-parameters for the baselines, which has not been found, and which could allow the baselines to perform better than DL models. Finally, many authors do not share their data or their code and therefore it is difficult to reproduce results across different papers. Thus, it is problematic to check whether a RS based on DL does really perform better than standard techniques. Among the 18 DL algorithms that they analyzed, only 7 were reproducible and even then with great effort. In addition to that, 6 of them can be outperformed by simpler models. For instance, they show how the NCF framework can be beaten by a simple linear model, called SLIM, developed by Ning and Karypis (2011).

This work adopts the viewpoint of Dacrema, Cremonesi, and Jannach (2019), agreeing that most studies that have demonstrated the superiority of DL techniques over standard ones relied on weak baselines and often overlooked tuning the baseline with a validation set. In particular, applying this same critical eye to Ahamed and Afroge (2019) and Howard and Gugger (2020)'s work, it is possible to identify some shortcomings. First, the authors tuned the number of epochs in the test-set and did not use a validation set, thus causing over-fitting. Then, NMF was tested against simple memory-based algorithms and a simple MF method. The same can be stated for the two functions developed in Fastai: Howard and Gugger (2020) claim that their results were state-of-the-art, however, they reached those results by executing a simple, educational experiment, only on the MF model and without a validation set. In addition to that, both experiments used algorithms that had not been trained to predict difficult cases where users have rated only one or two items, because in the MovieLens 100k dataset each user rates at least 20 movies.

2.1 Research question

The assumption behind this study is that, just as Dacrema, Cremonesi, and Jannach (2019) identified the methodological limitations of He et al. (2017)'s work, a more critical testing of the models developed by Howard and Gugger (2020) and Ahamed and Afroge (2019) should be conducted. The hypothesis is that such an accurate testing may show that a standard MF technique used as baseline, which has been properly tuned using a validation set, could have similar performances to DL enhanced techniques or, as an extreme case, perform even better. Given the aforementioned hypothesis, the research question is the following:

Can a properly tuned standard MF technique outperform recent DL-enhanced RSs based on explicit feedback?

In order to answer the research question, the MF Fastai, MLP Fastai and NMF model will be tested against the baseline on the Netflix dataset through two experiments. The experiments will follow specific conditions, to overcome the limitations identified in the testing process of Howard and Gugger (2020) and Ahamed and Afroge (2019):

1. a well-known and robust baseline for the Netflix dataset will be used, namely ImprovedRSVD developed by Paterek (2007);
2. different hyper-parameters will be tuned, in order to find the best configuration for the models.

The Netflix dataset will be used, because it presents more challenging data than the MovieLens 100k database: the cold start problem is highly present in the Netflix dataset, as a lot of users have rated only few movies, making it problematic for the RSs to make recommendations for those users (Wei et al. 2017). In the first experiment all the DL models will be compared with the baseline by tuning the number of epochs, the batch size and the learning rate. In the second experiment, to test the models' prediction performance even further, the DL models that make use of a MLP extension (MLP Fastai and NMF) will be tested by changing first the number of hidden layers and then the number of latent factors. A detailed explanation of the experiments can be found in section 3.2.5. These two experiments provide deeper insights into how DL models work and contribute to the debate that aims at comparing the effectiveness of DL vis-à-vis standard techniques.

3. Experimental Setup

This section describes the dataset and gives some information regarding the data processing. It then provides a description of the models and of the experiments' procedure. The code used to process the data and to perform the experiments was uploaded on GitHub and it is available for consultation at Lauria (2020).

3.1 Data

The two experiments have been carried out on the Netflix dataset, which was downloaded from Kaggle (Kaggle). This dataset was used in an open competition held by Netflix itself. The participants had to develop an algorithm which could beat Netflix's model to predict user ratings for movies. The grand prize of $1,000,000 was won by BellKor's Pragmatic Chaos team (Piotte and Chabbert 2009). The dataset contains over 100 million ratings from 480,000 randomly-chosen, anonymous Netflix customers over 17,000 movie titles. The data were collected between October 1998 and December 2005 and reflects the distribution of all ratings received during this period. The ratings are explicit feedback, as they are on a scale of 1 to 5 (integral) stars. To protect customer privacy, each customer ID has been replaced with a randomly-assigned ID. The date of each rating and the title and year of release for each movie ID are also provided. The training dataset comprises 4 different txt files, while the validation and test set are given in Probe.txt and Qualyfing.txt, respectively. The downloaded folder includes also a csv file, which lists the movie's ID, its year of release and its title.

Instead of using the entire dataset, only a sample of the whole 100 million ratings has been considered. Sampling the dataset is a practice common to many authors (e.g. Diop et al. (2019), Devooght and Bersini (2016) and Liang et al. (2018)), arguably because of limited access to hardware resources. As the temporal aspect is not the focus of this research, a random approach is the most efficient way to sample the whole dataset while maintaining the same target distribution and therefore keeping the imbalanced structure of the dataset. In the following paragraph, the data processing and some descriptive statistics will be provided, to offer a better understanding of the data.

3.1.1 Data processing. Given that CF approaches were implemented, the information required are: userID, movieID and rating. All the other features included in the dataset have not been considered. In addition to that, the Probe and Qualyfing data have not been used, because it is not possible to have access to the true ratings of the Qualyfing set anymore, and because the amount of ratings in the training data is already significant. The format of the training data that has been provided makes the data unsuitable to run the algorithms. The training data are divided into four different txt files and the first line of each file contains the movie ID. Each subsequent line in the four files corresponds to a rating from a customer in the following format: CustomerID, Rating. Thus data was processed to allow the models to learn. First, the format for each training .txt file was changed and then they were combined together in one new .csv file. In the new format CustomerID, MovieID and Rating are on the same line. At this stage, a random sample of 1 million ratings has been selected. The new dataset contains 289,901 users and 16,273 movies, which correspond to about 60% of users and 94% of movies compared to the whole Netflix dataset. 80% of this new dataset has been selected to form the training set and the remaining 20% has been used as test set. The training set has then been divided randomly again in order to have a validation set of 20%. Overall, this approach allows to keep the rating distribution constant, regardless of the amount of data.

3.1.2 Exploratory Data Analysis. As previously mentioned, this 1 million sample dataset is composed of 289,901 users and 16,273 movies, and the average rating is 3.6 on a scale of 1 to 5 stars (see Table(1) for other statistical data about rating). The distribution of all the ratings can be seen in Figure(2). Clearly, Netflix's users are quite generous in rating movies.

Table 1: Rating statistics

Statistics	**Value**
mean	3.6
std	1.1
min	1.0
25%	3.0
50%	4.0
75%	4.0
max	5.0

It is also interesting to note that, compared to the total number of users, the amount of ratings given by the majority of users is quite low, and the median is 2, which means that 50% of users have rated maximum two movies. This highlights the cold start problem in the given dataset and therefore the difficulty for RSs to predict ratings for users who have rated only one or two movies. In addition to that, the number of ratings that each movie has received is also quite low: 75% of movies have received 31 or less ratings and 50% of them only maximum 7 ratings. Evidently, there are few very popular movies that are rated by many users, while the majority of movies have received definitely less attention.

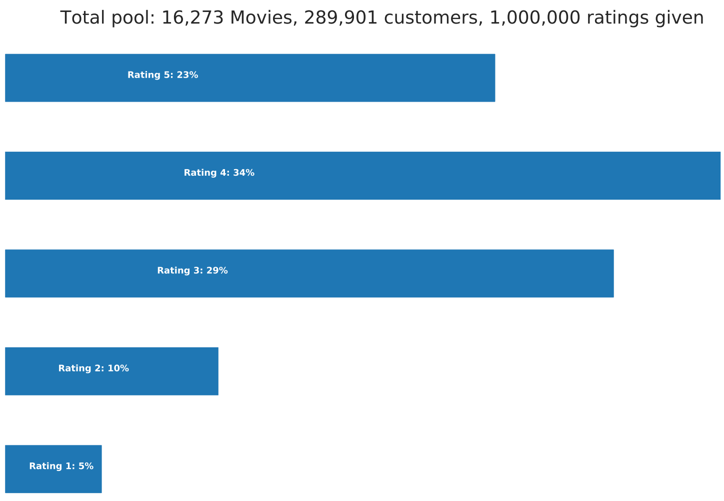


Figure 2: Rating distribution

In other studies (He et al. 2017), (Ahamed and Afroge 2019), (Diop et al. 2019), RSs algorithms were tested by removing all users that had fewer than some minimal number of ratings, or simply by using a standard dataset such as MovieLens, in order to overcome the cold start issue. The present work refrains from using such a simplistic approach, because it aims at gaining insights into RSs DL algorithms' behaviour with complex, imbalanced datasets.

3.2 Models and Methodology

This research relies on four models:

- Improved regularized SVD (ImprovedRSVD) described in Paterek (2007) (this one functions as baseline)
- MF Fastai implemented in the Fastai library (Howard and Gugger 2019)
- MLP Fastai implemented in the Fastai library (Howard and Gugger 2019)
- NMF described by Ahamed and Afroge (2019)

ImprovedRSVD has been selected as baseline because it previously proved to perform really well on the Netflix dataset (Paterek 2007), and can thus be considered as a very strong baseline in comparison to the memory-based models used in Ahamed and Afroge (2019). MF Fastai, MLP Fastai and NMF were selected among the most recent CF strategies based on DL, as they model directly the user-item matrix with explicit feedback. These models can be considered as neural extensions of the baseline, in as much as they replace the MF strategy with embedding layers and dense layers. The purpose of this study is to check whether a more simple standard technique could prove better than these three models, and, by doing so, to address the procedural limitations of the original experiments carried out by Ahamed and Afroge (2019) and Howard and Gugger (2020).

All the models were implemented using Python 3.6.9. Tab.(2) shows which libraries were used and their versions. As the author had limited GPU resources locally, Colab (Google) environment was used in order to benefit from its GPU to run DL models. The following sections (3.2.1, 3.2.2 and 3.2.3) will describe the selected algorithms and their implementation in Python. If it is not stated otherwise, default values were maintained for the arguments of the functions.

Table 2: Python libraries used to carry out the experiments

Library	**Version**
Fastai (Howard and Gugger 2019)	1.0.61
Numpy (Oliphant 2005)	1.18.4
Pandas (McKinney 2008)	1.0.3
Surprise (Hug 2015)	1.1.0
Tensorflow (Google 2015)	2.2.0
Keras (Chollet 2015)	2.3.1

3.2.1 Baseline. The *Surprise* library (Hug 2015) has been used to implement ImprovedRSVD. Surprise provides several functions that allow to develop MF-based RSs, memory-based RSs as well as clustering-based RSs. In order to implement ImprovedRSVD the function *SVD* was used. This function replicates the ImprovedRSVD prediction rule by default, as follows:

$$\widehat{r}_{ui} = \mu + b_i + b_u + p_u^T q_i \tag{1}$$

$\widehat{r}_{ui}$ denotes the rating of user u to the item i, p_u is the user latent feature vector and q_i is the item latent feature vector. b_u is the observed deviation of user u and b_i is the observed deviation of item i. μ is the average rating of all movies. The model learns by minimizing the following regularized squared error:

$$\sum_{r_{ui} \in R_{train}} (r_{ui} - \widehat{r}_{ui})^2 + \lambda(b_i^2 + b_u^2 + ||q_i||^2 + ||p_u||^2) \tag{2}$$

λ is the regularization term. Stochastic gradient descent is used to minimize the above error function.

3.2.2 Fastai models. The models described by Howard and Gugger (2020) are implemented in *Fastai* (Howard and Gugger 2019). The *EmbeddingDotBias* function calls the MF Fastai model. In order to predict ratings, MF Fastai first creates users, items and biases embeddings, then it calculates the dot product of embedded item and user, and then it adds the embedded biases. Finally, the resulting rating goes through a sigmoid layer, for which the range can be chosen, in order to introduce non-linearity into the output. According to the authors, this operation results in an improvement in prediction performance. Mathematically, the model can be expressed as:

$$\widehat{y}_{ui} = a_{out}(h^T((p_u q_i) + b_u + b_i)) \tag{3}$$

$\widehat{y}_{ui}$ indicates the predicted score of user u to the item i. a_{out} is the activation function, which in this case is a sigmoid, and h^T denotes the weights of the output layer. p_u and q_i are learnt with embedding layers as well as the biases b_u and b_i.

The MLP Fastai model can be called by using the *EmbeddingNN* function. In this case, the function replaces the dot product of the *EmbeddingDotBias* function with a MLP. ReLU is used as activation function for the hidden layers while the output layer is always a sigmoid function with the option to customize the range. Formally, the model can be described as:

$$\widehat{y}_{ui} = a_{out}(h^T \phi(z_{L-1})) \tag{4}$$

Where $\phi(z_{L-1}) = a_L(W^T z_{L-1} + b_L)a$ from 0 to L-1. Here, the weight matrix, bias vector and activation function are denoted by Wx, bx and ax respectively for the x-th layer's perceptron. For this project, the *Collablearner* Fastai function was used to call both models. The *usenn* argument of the *Collablearner* function establishes which model to call. As the *usenn* is False by default, the MF Fastai will be called. However, if it were set to True, the MLP Fastai would be used. The *Collablearner* function uses *Adam* optimizer by default to minimize the mean squared error, which is used as loss function.

3.2.3 NMF model. Unfortunately, it was not possible to find the code of the study carried out in Ahamed and Afroge (2019), therefore the model has been implemented from scratch by being as faithful as possible to the description provided by the authors in the paper. The function that replicates the NMF behaviour has been developed using Tensorflow and Keras layers. It was then implemented by using two distinct pairs of user and item embedding layers, one pair for the MF layer and the other one for the MLP layer. These pairs of embedding layers are trained separately according to Eq.(4) for the MLP layer and to the following equation for the MF layer:

$$\widehat{y}_{ui} = \sum_{k=1}^{k} p_{uk} q_{ik} \tag{5}$$

p_u and q_i indicates the embedding for user u and item i, respectively. K denotes the number of latent features for the embeddings.

After that, they are combined with the *concatenate* layer and this combination goes first to a dense layer and then to the final output layer, which, according to the description, always uses ReLU as activation function. Also in this case *Adam* is relied upon to minimize the mean squared error, which is used as loss function. NMF layer can be written as:

$$\widehat{y}_{ui} = a_{out}\left(h^T a\left(p_u q_i + W \begin{bmatrix} p_u \\ q_i \end{bmatrix} + b\right)\right) \tag{6}$$

3.2.4 Metrics. In the majority of the studies carried out using the Netflix dataset, such as Koren (2010), Paterek (2007), Hong and Tsamis (2006) and many others, the quality of the results is measured by their Root Mean Squared Error (RMSE). Therefore, this project too relies on the RMSE to compare algorithms' prediction performance. Formally, RMSE is expressed as:

$$\sqrt{\sum_{(u,i)\in TestSet} \frac{(r_{ui} - \widehat{r}_{ui})^2}{|TestSet|}} \tag{7}$$

3.2.5 Experiments. To answer the research question *"can a properly tuned standard MF technique outperform recent DL-enhanced RSs based on explicit feedback?"*, the models were tested and their performance was compared in two experiments. Fig.(3) gives an overview of which models were tested in which experiment, and what operations were carried out. The Green rectangles indicate tuning or re-tuning phases and specify hyper-parameters that get tested in each case; the blue rectangles indicate the testing phases, meaning that the RMSE is computed on the test set to see which model performs best. In the second experiment, only MLP Fastai and NMF are considered and tuned for a second time. The yellow diamond shows that if improvement is registered, the models are tested again on the test set and compared to the baseline. As opposed to what was done by both Ahamed and Afroge (2019) and Howard and Gugger (2020), this project tries to tune many hyper-parameters throughout the two experiments in order to identify as many top performing hyper-parameters as possible, given the current dataset.

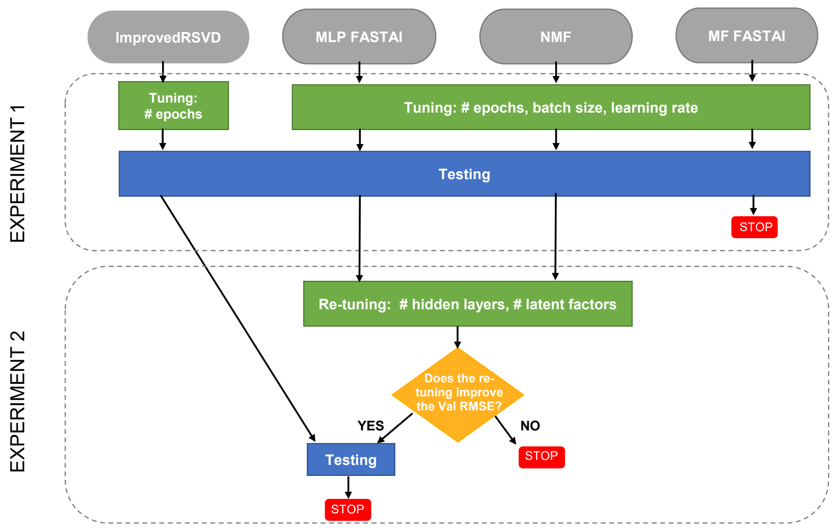

ImprovedRSVD
MLP FASTAI
NMF
MF FASTAI
EXPERIMENT 1
Tuning: # epochs
Tuning: # epochs, batch size, learning rate
Testing
STOP
EXPERIMENT 2
Re-tuning: # hidden layers, # latent factors
Does the re-tuning improve the Val RMSE?
YES
NO
Testing
STOP
STOP

Figure 3: Workflow chart of the two experiments

The goal of the first experiment is to test whether MF Fastai, MLP Fastai and NMF can perform better than a standard technique by tuning the number of epochs for the baseline, and the number of epochs, batch size and learning rate for the DL models. The number of latent factors (=10) for all the algorithms, as well as the number of hidden layers (=3) for the MLP Fastai and NMF, have not been tuned. In the first experiment the number of latent factors (=10) was chosen as an attempt to strengthen the baseline. That is because SVD proved to perform better in 10 dimensions than in 30 (Mnih and Salakhutdinov 2008), the author assumed that ImprovedRSVD (the baseline of this study) would behave accordingly. However, to avoid a bias, in the second experiment different numbers of latent factors will be tested for MLP Fastai and NMF, which will then be tested against the baseline. In order to tune the algorithms, different values of batch size, learning rate and number of neurons per hidden layers must be identified. Instead of choosing arbitrarily, the author relied on the choice of values found in He et al. (2017). More in details, batch size = [128, 256, 512, 1024] and learning rate = [0.0001, 0.0005, 0.001, 0.005] have been tested for MF Fastai and NMF. Given that MLP Fastai can be considered as an extension of MF Fastai, it has been assumed that the best batch size and learning rate for MF Fastai would also be suitable for MLP Fastai. As a result, tuning time decreased considerably. In addition to that, also some arbitrary numbers of units for the three hidden layers have been tuned, namely [256, 128, 64], [128, 64, 32], [64, 32, 16] and [32, 16, 8]. In the case of the first vector, this means that the first hidden layer has 256 units, the second 128 and the third 64. The same interpretation holds true for the other vectors too. The final RMSE is computed by using the best set of parameters found during the tuning stage. The algorithms are run on the test set which is the same for all the algorithms, while the split between train and validation sets is computed separately for each algorithm but with the same proportion: 80% train and 20% val.

In the second experiment both NMF and Fastai MLP are tested, first with different numbers of hidden layers, and then with a different number of latent factors in order to see how these hyper-parameters affect their prediction performance against the baseline. For both DL models, the RMSE is first calculated on the validation set, and if the results show improvement compared to the corresponding val RMSE of the first experiment, then the RMSE is calculated also on the test set. Particularly, number of latent factors = [20, 30, 40, 50, 60, 70, 80, 90, 100] and number of hidden layers = [1, 2, 3, 4, 5, 6, 7] have been tested. As the number of hidden layers increases, the number of units per layer increases too. For example, when only one hidden layer is tested, the number of units per layer is 8, when two hidden layers are tested, the number of units becomes [16, 8], which means that the first hidden layer has 16 units, while the second hidden layer has 8 units. With three hidden layers the units become [32, 16, 8] and so on until 7 hidden layers are tested (the number of units then becomes [512, 256, 128, 64, 32, 16, 8]). The values of batch size and learning rate were not changed, they correspond to the best values found in experiment one. For the Fastai MLP, batch size = 256 and learning rate = 0.0005; whereas batch size = 256 and learning rate = 0.0001 for the NMF model. When the number of factors changes, the number of hidden layers is maintained at three for both models with units [256, 128, 68]. When the number of hidden layers changes, the number of latent factor is kept at 10.

4. Results

In this section results are provided. Paragraph 4.0.1 describes the results of the first experiment and paragraph 4.0.2 describes the results of the second experiment.

4.0.1 Results of experiment 1. This section will present the results of MF Fastai, MLP Fastai and NMF prediction performance on the Netflix dataset. During the tuning stage, the models were run as described in section 3.2.5, and a set of parameters was identified, which proved to perform better than the others. Table (3) illustrates the results of the first experiment. The first column shows the name of the model, the second column presents the parameters and the third column highlights the test RMSE. The models are ordered by test RMSE value, from the lowest to the highest: ImprovedRSVD has the lowest RMSE (0.98) while NMF has the highest (1.03). The most simple model, namely ImprovedRSVD, proved to be the best performer, as it achieved the lowest RMSE value. Among the three DL models, the Fastai models proved to perform better than NMF, and particularly MLP Fastai reaches closest to the baseline.

Table 3: Results of the first experiment. For each model, the name, the parameters and the test RMSE are provided. The models are put in order from the lowest RMSE to the highest. The three DL models analyzed do not pass the baseline.

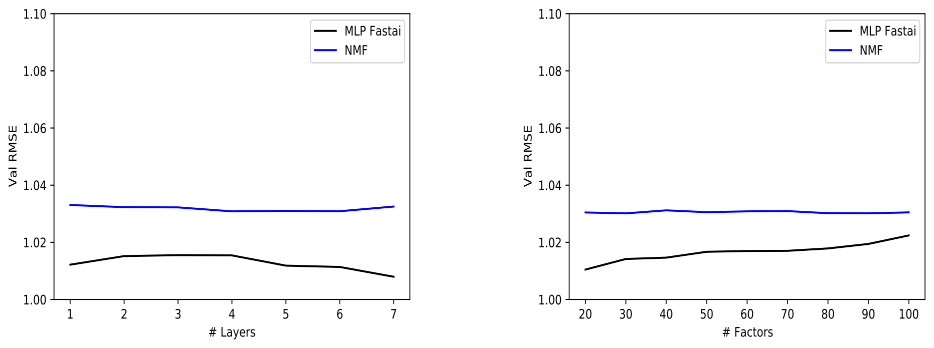
Model	Parameters	Test RMSE
ImprovedRSVD	Epochs = 20	**0.98**

MLP Fastai	Epochs = 2
	Batch size = 256
	Learning rate = 0.0005
	Layers = [256, 128, 64]
		1.00

MF Fastai	Epochs = 8 Batch size = 256 Learning rate = 0.0005	1.02

NMF	Epochs = 5
	Batch size = 256
	Learning rate = 0.0001
	Layers = [256, 128, 64]
		1.03

4.0.2 Results of experiment 2. Experiment 2 completes the first experiment, in as much as it tunes two hyper-parameters that were not considered in the first experiment. MLP Fastai and NMF were tuned again, first with different numbers of hidden layers and then, with a different number of latent factors. If improvement was registered, the models would be compared again to the baseline. Fig.(4) shows how the validation RMSE changes as the number of hidden layers (4a) and the number of latent factors (4b) increase. In both figures, (4a) and (4b), the NMF model is depicted in blue, while the MLP Fastai model is depicted in black. The y-axis shows the val RMSE and the x-axis shows either the number of layers (4a) or the number of latent factors (4b).



(a) Effect of number of factors to the val RMSE (b) Effect of the number of layers to the val RMSE

Figure 4: Fig.(4a) shows how the val RMSE changes according to changes in the number of hidden layers. Fig.(4b) shows how the val RMSE changes according to changes in the number of factors. In both figures, the NMF model is depicted in blue, while the MLP Fastai model is depicted in black.

First of all, it is clear that even after changing these parameters, neither model has a val RMSE lower than 1.0. It is also possible to see that the NMF performance is unaffected both in Figure (4a) and (4b): the val RMSE has a fixed value of 1.03. As no sign of improvement was shown in the validation set, the NMF was not run on the test set. On the other hand, these parameters have a marginal effect on the MLP Fastai model. As shown in Fig.(4a), the val RMSE increases as the number of hidden layer increases, until it reaches a value of 4, after which it starts to decrease until it reaches almost 1.00. Fig(4b) clearly shows that as the number of latent factors increases, so does the val RMSE: its value starts from 1.01 with a latent factors number of 20 and it increases to reach 1.02 with a latent factors number of 100.

Because of the improvement caused by adding hidden layers, the MLP Fastai was run also on the test-set with seven hidden layers, which corresponds to its lowest RMSE value, while the number of latent factors was not changed because the validation set showed that if it were changed, the model would actually perform worse. This subsequent test resulted in a test RMSE value of 1.00 for the MLP Fastai. That means that there was no variation compared to the first experiment, and therefore further tuning did not improve algorithm performance.

5. Discussion

This research intended to verify whether DL models are to be preferred as a rule over standard techniques to develop RSs when modelling explicit feedback. Even though a plethora of studies exist, that show promising results when using RSs based on DL (Van den Oord, Dieleman, and Schrauwen 2013), (Xu et al. 2016), (He et al. 2017) and (Cao, Yang, and Liu 2017), Dacrema, Cremonesi, and Jannach (2019) proved that DL is not necessarily better than standard techniques, and that by carrying out accurate and precise experiments, results may vary. This issue has practical implications, because if standard techniques could prove to perform better than DL based RSs, then companies could utilize more simple standard techniques, instead of investing in DL algorithms. Aligning with Dacrema, Cremonesi, and Jannach (2019), this work agrees that most studies that have demonstrated the superiority of DL techniques over standard ones relied on weak baselines and often overlooked tuning the baseline with a validation set. Particularly this work identified some limitations of three recent DL based algorithms, namely MF Fastai, MLP Fastai and NMF, developed by Howard and Gugger (2020) and Ahamed and Afroge (2019) using the NCF framework by He et al. (2017). This work suggests that these studies used overly simple, balanced datasets to test algorithms; neither of them used a validation set, relying only on the test set and therefore potentially causing overfitting; Ahamed and Afroge (2019)'s algorithms were tested against simple memory-based RSs, instead of more robust model-based techniques; and Howard and Gugger (2020)'s MLP was never tested in a scientific work. Formally, the research question can be expressed as follows:

> **Can a properly tuned standard MF technique outperform recent DL-enhanced RSs based on explicit feedback?**

The research question has been answered by carrying out two experiments under the following specific conditions, to overcome the limitations identified in the testing process of Howard and Gugger (2020) and Ahamed and Afroge (2019). First, ImprovedRSVD was chosen as baseline, because it is one of the strongest standard methods for the Netflix dataset in comparison with memory-based RSs, as evidenced by Koren, Bell, and Volinsky (2009). Then, the Netflix dataset was used, because it presents the cold start problem, as opposed to MovieLens 100k database. Finally, different hyperparameters were tuned, in order to find the best configuration for both the baseline and the DL models.

In Experiment 1, MF Fastai, MLP Fastai, NMF and the baseline were first tuned by trying different numbers of epochs, batch size and learning rate. MLP Fastai and NMF were also tuned with three hidden layers while number of units per layer varied. The best set of parameters was identified and used to test the models on the test set. The baseline came first, scoring 0.98 in terms of RMSE on the test data, followed by MLP Fastai, which was tested in the present work for the first time and scored 1.00. MF Fastai scored 1.02, and finally NMF scored 1.03, ending up last. The results of the first experiment show that the baseline, ImprovedRSVD, was the top performer, as it achieved the lowest RMSE value.

Wishing to take the testing a step further, a second experiment was carried out, in which NMF and MLP Fastai were tuned again, this time changing the number of latent factors and the number of hidden layers. As shown in Fig.(4), changing these parameters did not have an effect on the NMF model but it had a marginal effect on the MLP Fastai model. When the number of hidden layers is higher than 4, the RMSE decreases from 1.01 to 1.00. On the other hand, when the number of latent factors increases, the RMSE increases too. The MLP Fastai was also run on the test-set with seven hidden layers, which corresponds to its lowest RMSE value, while the number of latent factors was not changed. This subsequent test resulted in a test RMSE value of 1.00 for the MLP Fastai. That means that there was no variation compared to the first experiment, and therefore further tuning did not improve algorithm performance.

The results of these two experiments provide sufficient evidence to support the validity of the initial hypothesis and answer the research question, which was: *"can a properly tuned standard MF technique outperform recent DL-enhanced RSs based on explicit feedback?"*. The experiments show that MF Fastai, MLP Fastai and NMF do not perform better than simpler model-based RSs, when a more rigorous procedure is applied, than the one used by Ahamed and Afroge (2019) and Howard and Gugger (2020). Quite the opposite, the ImprovedRSVD performed consistently better, not only when all models were tuned the first time, but even after MLP Fastai and NMF were tuned a second time to improve them. These findings are significant to the general discussion related to the alleged superiority of DL based RSs over standard methods, for explicit feedback datasets while also supporting and expanding on the findings of Dacrema, Cremonesi, and Jannach (2019).

An interesting observation that can be made on the results of the second experiment is that calculating MLP Fastai's test RMSE with seven hidden layers, yields exactly the same value (test RMSE = 1.00) as testing it with three hidden layers. However, MF Fastai, which can be considered as MLP Fastai with zero hidden layers, proved to perform worse on the test than MLP Fastai with three hidden layers (see Table (3)). It is reasonable to say that test RMSE improves when the number of hidden layers increases up to 3. However, once the number of hidden layers surpasses this threshold, the test RMSE does not improve anymore. Therefore, this research supports the findings of He et al. (2017), Wu et al. (2016), who stated that increasing the number of hidden layers the accuracy improves, but differs from their conclusion in as much as it sets a threshold of maximum 3 hidden layers, beyond which the accuracy does not improve anymore.

Finally, it is worth focusing on some shortcomings of this study. First, although, this project tried to tune as many hyper-parameters as possible, not all of them were considered because of time restraint. For instance, the weight decay in Fastai (Howard and Gugger 2019) has not been tuned, it was left with its default value (0.01). The second limitation was that, no Dropout was used for the DL models. Finally, both the Fastai models and the NMF model, were implemented using a simple Mean Absolute Error as loss function, while the ImprovedRSVD implemented with Surprise (Hug 2015) has by default a more complex loss function that considers also the biases for users and items. It may not be worth correcting the first limitation of this study. As the most important hyper-parameters were already tuned in this project, further changes may not yield a different result. The same can be stated also for the second limitation, because the author used the EarlyStopping technique to fight overfitting. Therefore, even if a different technique to fight overfitting was used, the results may not be affected. On the other hand, it would be worth it to correct the third shortcoming and to implement a stronger loss function for the DL models, which would consider also biases for users and items. Changing the loss function could effectively change the results of the DL models. Hopefully future research will address this issue.

6. Conclusion

Starting from the methodological approach of Dacrema, Cremonesi, and Jannach (2019), this work identified and addressed the limitations of recent DL based models for explicit feedback: MLP Fastai and MF Fastai, created by Howard and Gugger (2020); and NMF created by Ahamed and Afroge (2019). This study aimed at finding an answer to the research question: *can a properly tuned standard MF technique outperform recent DL-enhanced RSs based on explicit feedback?* To answer this question, unlike the original studies, first, a more challenging dataset (the Netflix dataset) was used, and a strong baseline for the given dataset (ImprovedRSVD) was selected. Then, the models were tuned with different configurations of hyper-parameters in two consecutive experiments. The experiments showed that the ImprovedRSVD performed consistently better than the DL models, not only when all models were tuned the first time, but even after different numbers of hidden layers and latent factor were tuned, to improve MLP Fastai and NMF even further. The results showed that the baseline reached a test RMSE of 0.98 while the best DL model reached a test RMSE of 1.00. These findings effectively answer the research question and are significant to the general academic debate related to the alleged superiority of DL based RSs over standard methods, for explicit feedback datasets, while also supporting and expanding on the findings of Dacrema, Cremonesi, and Jannach (2019). As mentioned in the Discussion section, one limitation of this work that should be addressed in the future is that it used a simpler loss function for the DL models compared to the more complex loss function used to implement the baseline, which includes also the biases for users and items. Future research will hopefully address this limitation and implement a stronger loss function for the DL models.

References

Adomavicius, Gediminas and Alexander Tuzhilin. 2005. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE transactions on knowledge and data engineering*, 17(6):734–749.

Ahamed, M. T. and S. Afroge. 2019. A recommender system based on deep neural network and matrix factorization for collaborative filtering. In *2019 International Conference on Electrical, Computer and Communication Engineering (ECCE)*, pages 1–5.

Cao, Sanxing, Nan Yang, and Zhengzheng Liu. 2017. Online news recommender based on stacked auto-encoder. In *2017 IEEE/ACIS 16th International Conference on Computer and Information Science (ICIS)*, pages 721–726, IEEE.

Chollet, François. 2015. Keras guide. https://keras.io/guides/.

Collobert, Ronan and Jason Weston. 2008. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pages 160–167.

Dacrema, Maurizio Ferrari, Paolo Cremonesi, and Dietmar Jannach. 2019. 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.

Devooght, Robin and Hugues Bersini. 2016. Collaborative filtering with recurrent neural networks. *arXiv preprint arXiv:1608.07400*.

Diop, Mamadou, Sebastian Miron, Anthony Larue, and David Brie. 2019. Binary matrix factorization applied to netflix dataset analysis. *IFAC-PapersOnLine*, 52(24):13–17.

Funk, Simon. 2006. Netflix update: Try this at home. https://sifter.org/~simon/journal/20061211.html. Accessed: 2020-04-10.

Google. Welcome to colaboratory. https://colab.research.google.com/notebooks/intro.ipynb. Accessed: 2020-05-5.

Google. 2015. Tensorflow guide. https://www.tensorflow.org/guide.

He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016a. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

He, Xiangnan, 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*, WWW '17, page 173–182, International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, CHE.

He, Xiangnan, Hanwang Zhang, Min-Yen Kan, and Tat-Seng Chua. 2016b. Fast matrix factorization for online recommendation with implicit feedback. In *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval*, pages 549–558.

Hill, Will, Larry Stead, Mark Rosenstein, and George Furnas. 1995. Recommending and evaluating choices in a virtual community of use. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 194–201.

Hong, Richang, Zhenzhen Hu, Luoqi Liu, Meng Wang, Shuicheng Yan, and Qi Tian. 2015. Understanding blooming human groups in social networks. *IEEE Transactions on Multimedia*, 17(11):1980–1988.

Hong, Ted and Dimitris Tsamis. 2006. Use of knn for the netflix prize. *CS229 Projects*.

Howard, J and S.Gugger. 2019. Collaborative filtering example. https://github.com/fastai/course-v3/blob/master/nbs/dl1/lesson4-collab.ipynb. Accessed: 2020-05-4.

Howard, Jeremy and Sylvain Gugger. 2019. Fastai documentation. https://docs.fast.ai/. Accessed: 2020-04-25.

Howard, Jeremy and Sylvain Gugger. 2020. Fastai: A layered api for deep learning. *Information*, 11(2):108.

Hug, Nicolas. 2015. Surprise documentation. https://surprise.readthedocs.io/en/stable/index.html. Accessed: 2020-04-25.

Kaggle. Kaggle netflix data. https://www.kaggle.com/netflix-inc/netflix-prize-data. Accessed: 2020-05-1.

Karatzoglou, Alexandros and Balázs Hidasi. 2017. Deep learning for recommender systems. In *Proceedings of the Eleventh ACM Conference on Recommender Systems*, RecSys '17, page 396–397, Association for Computing Machinery, New York, NY, USA.

Koren, Yehuda. 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*, pages 426–434.

Koren, Yehuda. 2010. Factor in the neighbors: Scalable and accurate collaborative filtering. *ACM Trans. Knowl. Discov. Data*, 4(1).

Koren, Yehuda, Robert Bell, and Chris Volinsky. 2009. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37.

Lauria, Francesco. 2020. Does dl really improve standard cf strategies? `https://github.com/F-Lauria/Deep_CF_Netflix`.

Li, Sheng, Jaya Kawale, and Yun Fu. 2015. Deep collaborative filtering via marginalized denoising auto-encoder. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, CIKM '15, page 811–820, Association for Computing Machinery, New York, NY, USA.

Lian, Jianxun, Fuzheng Zhang, Xing Xie, and Guangzhong Sun. 2017. Cccfnet: A content-boosted collaborative filtering neural network for cross domain recommender systems. In *Proceedings of the 26th international conference on World Wide Web companion*, pages 817–818.

Liang, Dawen, Rahul G. Krishnan, Matthew D. Hoffman, and Tony Jebara. 2018. Variational autoencoders for collaborative filtering. In *Proceedings of the 2018 World Wide Web Conference*, WWW '18, page 689–698, International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, CHE.

McKinney, Wes. 2008. Pandas documentation. `https://pandas.pydata.org/docs/`.

Mehta, R. and K. Rana. 2017. A review on matrix factorization techniques in recommender systems. In *2017 2nd International Conference on Communication Systems, Computing and IT Applications (CSCITA)*, pages 269–274.

Mnih, Andriy and Russ R Salakhutdinov. 2008. Probabilistic matrix factorization. In *Advances in neural information processing systems*, pages 1257–1264.

Netflix. 2006. Netflix prize. `https:https://www.netflixprize.com/`. Accessed: 2020-05-2.

Ning, Xia and George Karypis. 2011. Slim: Sparse linear methods for top-n recommender systems. In *2011 IEEE 11th International Conference on Data Mining*, pages 497–506, IEEE.

Oliphant, Travis. 2005. Numpy documentation. `https://numpy.org/doc/stable/`.

Van den Oord, Aaron, Sander Dieleman, and Benjamin Schrauwen. 2013. Deep content-based music recommendation. In *Advances in neural information processing systems*, pages 2643–2651.

Patel, B., P. Desai, and U. Panchal. 2017. Methods of recommender system: A review. In *2017 International Conference on Innovations in Information, Embedded and Communication Systems (ICIIECS)*, pages 1–4.

Paterek, Arkadiusz. 2007. Improving regularized singular value decomposition for collaborative filtering. In *Proceedings of KDD cup and workshop*, volume 2007, pages 5–8.

Piotte, Martin and Martin Chabbert. 2009. The pragmatic theory solution to the netflix grand prize. *Netflix prize documentation*.

Ranjan, Ankur A, Amod Rai, Saiful Haque, Bhanu P Lohani, and Pradeep K Kushwaha. 2019. An approach for netflix recommendation system using singular value decomposition. *Journal of Computer and Mathematical Sciences*, 10(4):774–779.

Rendle, Steffen, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2012. Bpr: Bayesian personalized ranking from implicit feedback. *arXiv preprint arXiv:1205.2618*.

Resnick, Paul, Neophytos Iacovou, Mitesh Suchak, Peter Bergstrom, and John Riedl. 1994. Grouplens: An open architecture for collaborative filtering of netnews. In *Proceedings of the 1994 ACM Conference on Computer Supported Cooperative Work*, CSCW '94, page 175–186, Association for Computing Machinery, New York, NY, USA.

Sarwar, Badrul, George Karypis, Joseph Konstan, and John Riedl. 2000. Application of dimensionality reduction in recommender system-a case study. Technical report, Minnesota Univ Minneapolis Dept of Computer Science.

Shardanand, Upendra and Pattie Maes. 1995. Social information filtering: algorithms for automating "word of mouth". In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 210–217.

Sharma, Amit, Jake M Hofman, and Duncan J Watts. 2015. Estimating the causal impact of recommendation systems from observational data. In *Proceedings of the Sixteenth ACM Conference on Economics and Computation*, pages 453–470.

Shi, Yue, Martha Larson, and Alan Hanjalic. 2014. Collaborative filtering beyond the user-item matrix: A survey of the state of the art and future challenges. *ACM Comput. Surv.*, 47(1).

Singhal, Ayush, Pradeep Sinha, and Rakesh Pant. 2017. Use of deep learning in modern recommendation system: A summary of recent works. *arXiv preprint arXiv:1712.07525*.

Smith, Brent and Greg Linden. 2017. Two decades of recommender systems at amazon. com. *Ieee internet computing*, 21(3):12–18.

Vasile, Flavian, Elena Smirnova, and Alexis Conneau. 2016. Meta-prod2vec: Product embeddings using side-information for recommendation. In *Proceedings of the 10th ACM Conference on Recommender Systems*, pages 225–232.

Vuurens, Jeroen BP, Martha Larson, and Arjen P de Vries. 2016. Exploring deep space: Learning personalized ranking in a semantic space. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*, pages 23–28.

Wang, Xiang, Xiangnan He, Liqiang Nie, and Tat-Seng Chua. 2017. Item silk road: Recommending items from information domains to social users. In *Proceedings of the 40th International ACM SIGIR conference on Research and Development in Information Retrieval*, pages 185–194.

Wei, Jian, Jianhua He, Kai Chen, Yi Zhou, and Zuoyin Tang. 2017. Collaborative filtering and deep learning based recommendation system for cold start items. *Expert Systems with Applications*, 69:29–39.

Wu, Yao, Christopher DuBois, Alice X Zheng, and Martin Ester. 2016. Collaborative denoising auto-encoders for top-n recommender systems. In *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining*, pages 153–162.

Xu, Zhenghua, Cheng Chen, Thomas Lukasiewicz, Yishu Miao, and Xiangwu Meng. 2016. Tag-aware personalized recommendation using a deep-semantic similarity model with negative sampling. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, pages 1921–1924.

Xue, Hong-Jian, Xinyu Dai, Jianbing Zhang, Shujian Huang, and Jiajun Chen. 2017. Deep matrix factorization models for recommender systems. In *IJCAI*, pages 3203–3209.

Zhang, Shuai, Lina Yao, Aixin Sun, and Yi Tay. 2019. Deep learning based recommender system: A survey and new perspectives. *ACM Computing Surveys (CSUR)*, 52(1):1–38.

Zhang, Shuai, Lina Yao, Aixin Sun, Sen Wang, Guodong Long, and Manqing Dong. 2018. Neurec: On nonlinear transformation for personalized ranking. *arXiv preprint arXiv:1805.03002*.