Representation Learning and Pairwise Ranking for Implicit and Explicit Feedback in Recommendation Systems

Mikhail Trofimov
Dorodnitsyn Computing Center of Russian Academy of Sciences
Moscow, Russia
mikhail.trofimov@phystech.edu

Sumit Sidana
LIG, Université de Grenoble Alpes
Grenoble, France
sumit.sidana@imag.fr

Oleh Horodnitskii
CES, Skolkovo Institute of Science and Technology
Moscow, Russia
o.gorodnitskii@skoltech.ru

Charlotte Laclau
LIG, Université de Grenoble Alpes
Grenoble, France
charlotte-laclau@univ-grenoble-alpes.fr

Yury Maximov*
CES, Skolkovo Institute of Science and Technology, and Theoretical Division T-4 & CNLS, Los Alamos National Laboratory
Los Alamos, NM 87544
yury@lanl.gov

Massih-Reza Amini
LIG, Université de Grenoble Alpes
Grenoble, France
massih-reza.amini@imag.fr

ABSTRACT

In this paper, we propose a novel ranking approach for collaborative filtering based on Neural-Networks that jointly learns a new representation of users and items in an embedded space as well as the preference relation of users over pairs of items. The learning objective is based on two ranking losses that control the ability of the model to respect the ordering over the items induced from the users preferences, as well as, the capacity of the dot-product defined in the learned embedded space to produce the ordering. The proposed model is by nature suitable for both implicit and explicit feedback and involves the estimation of only very few parameters. Through extensive experiments on several real-world benchmarks, both explicit and implicit, we show the interest of learning the preference and the embedding simultaneously when compared to learning those separately. We also demonstrate that our approach is very competitive with the best state-of-the-art collaborative filtering techniques proposed independently for explicit and implicit feedback.

CCS CONCEPTS

• **Information systems** → **Learning to rank**; **Recommender systems**; *Language models*;

KEYWORDS

Recommender Systems; Neural Networks; Collaborative Filtering

*Corresponding author

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 123-4567-24-567/08/06…$15.00
https://doi.org/10.475/123_4

ACM Reference format:
Mikhail Trofimov, Sumit Sidana, Oleh Horodnitskii, Charlotte Laclau, Yury Maximov, and Massih-Reza Amini. 2017. Representation Learning and Pairwise Ranking for Implicit and Explicit Feedback in Recommendation Systems. In *Proceedings of ACM Conference, Washington, DC, USA, July 2017 (Conference'17),* 11 pages.
https://doi.org/10.475/123_4

1 INTRODUCTION

In the recent years, recommender systems (RS) have attracted many interest in both the industry and the academic research community, mainly due to the new challenges that the design of a decisive and scalable RS present. Given a set of customers (or users), the goal of RS is to provide a personalized recommendation of products to users who would likely be of their interest. Common examples of applications include the recommendation of movies (Netflix, Amazon Prime Video), music (Pandora), videos (Youtube), news content (Outbrain) or advertisements (Google). The development of efficient RS is critical both from the company and the consumer perspectives. Indeed, on one hand, users usually face a very large number of options: for instance, Amazon proposes over 20,000 movies in its selection, and it is therefore important to help them to take the best possible decision. On the other hand, major companies report significant increase of their traffic or sales coming from personalized recommendations: Amazon declares that 35% of its sales is generated by recommendations, two-thirds of the movies watched on Netflix are recommended and 28% of ChoiceStream users said that they would buy more music, provided the fact that they meet their tastes and interests.[1]

Two main approaches have been proposed to tackle this problem [28]. The first one, referred to as Content-Based recommendation technique [23, 26] make use of existing contextual information about the users (e.g. demographic information) or items (e.g. textual description) for recommendation. The second approach, referred to as collaborative filtering (CF) and undoubtedly the most popular

[1] Talk of Xavier Amatriain - Recommender Systems - Machine Learning Summer School 2014 @ CMU.

one, relies on the past interactions and recommends items to users based on the feedback provided by similar other users. Feedback can be *explicit*, provided mostly by ratings; or *implicit* that include clicks, browsing over an item or listening to a song. Such implicit feedback is readily available in abundance but is more challenging to take into account as it does not clearly translate the preference of a user for an item. The adaptation of CF systems designed for one type of feedback to another has shown to be suboptimal as the basic hypothesis of these systems inherently depends on the nature of the feedback [38]. Further, the dyadic representation of users and items has shown to be the bottleneck of these systems [36], mostly in the case where contextual information over users and items allowing to have a richer representation is unavailable.

In this paper we propose to address the following questions : (Q_1) How to design a CF model that deals with implicit and explicit feedback? And, (Q_2) how to learn both efficient and reliable representations for CF without using any contextual information? To do so, we propose a Neural-Network based CF model that simultaneously learns the preference of users over pairs of items, as well as their representations in an embedded space. The learning objective is a double ranking loss, where the first one directs the model to learn the preferences of users over the pairs of items, and the second one controls that the dot product function in the learned embedded space also respects this preference.

The remainder of this paper is organized as follows. Section 2 provides an overview of existing related methods. Section 3 defines the proposed pairwise ranking approach and the optimization procedure. Section 4 is devoted to numerical experiments on six real-world benchmarks including MovieLens, Netflix, and two implicit datasets build from challenges. We compare our model with state- of-the-art methods showing the appropriateness of our contribution. Finally we summarize the study and give possible future research perspectives in the last section.

2 RELATION TO STATE-OF-THE-ART

This section provides an overview of the state-of-the-art approaches that are the most similar to ours.

2.1 Neural Language Models

Neural language models have proven themselves successful in many natural language processing tasks including speech recognition, information retrieval and sentiment analysis. These models are based on a distributional hypothesis stating that words, occurring in the same context with the same frequency, are similar. In order to capture such similarities, these approaches propose to embed the word distribution into a low-dimensional continuous space using Neural Networks, leading to the development of several powerful and highly scalable language models such as the word2Vec Skip-Gram (SG) model [24, 25, 32].

The recent work of [19] has shown new opportunities to extend the word representation learning to characterize more complicated piece of information. In fact, the authors of this paper established the equivalence between SG model with negative sampling, and implicitly factorizing a point-wise mutual information (PMI) matrix. Further, they demonstrated that word embedding can be applied to different types of data, thereby proving that it is possible to design an appropriate context matrix for them. This idea has been successfully applied to recommendation systems where different approaches attempted to learn representations of items and users in an embedded space in order to achieve more efficiently the problem in hand, compared to the traditional case where these representations are not learned [11, 12, 21]. In [21], the authors proposed a model that relies on the intuitive idea that pairs of items scored the same way by different users are similar. The approach reduces to finding both the latent representations of users and items, with the traditional Matrix Factorization (MF) approach, and simultaneously learning item embeddings using a co-occurrence shifted positive PMI (SPPMI) matrix defined by items and their context. The latter is used as a regularization term in the traditional objective function of MF. Similarly, in [11] the authors proposed Prod2Vec, which embeds items using a neural language model applied to a time series of user purchases. This model was further extended in [35] who, by defining appropriate context matrices, proposed Meta-Prod2Vec. Their approach learns a representation for both items and side information available in the system. The embedding of additional information is further used to regularize the item embedding. Inspired by the concept of sequence of words; the approach proposed by [12] defined the consumption of items by users as trajectories. Then, the embedding of items is learned using the SG model and the users embedding is further inferred as to predict the next item in the trajectory. In these approaches, the learning of item and user representations are employed to make prediction with predefined or fixed similarity functions (such as dot-products) in the embedded space.

2.2 Learning-to-Rank with Neural Networks

Motivated by automatically tuning the parameters involved in the combination of different scoring functions, Learning-to-Rank approaches were originally developed for Information Retrieval (IR) tasks and are grouped into three main categories: pointwise, listwise and pairwise [22].

Pointwise approaches [9, 20] assume that each queried document pair has an ordinal score. Ranking is then formulated as a regression problem, in which the rank value of each document is estimated as an absolute quantity. In the case where relevance judgments are given as pairwise preferences (rather than relevance degrees), it is usually not straightforward to apply these algorithms for learning. Moreover, pointwise techniques do not consider the inter dependency among documents, so that the position of documents in the final ranked list is missing in the regression-like loss functions used for parameter tuning. On the other hand, listwise approaches [39, 40] take the entire ranked list of documents for each query as a training instance. As a direct consequence, these approaches are able to differentiate documents from different queries, and consider their position in the output ranked list at the training stage. Listwise techniques aim to directly optimize a ranking measure, so they generally face a complex optimization problem dealing with non-convex, non-differentiable and discontinuous functions. Finally, in pairwise approaches [8, 10, 16] the ranked list is decomposed into a set of document pairs. Ranking is therefore considered as the classification of pairs of documents, such that a classifier is trained

by minimizing the number of misorderings in ranking. In the test phase, the classifier assigns a positive or negative class label to a document pair that indicates which of the documents in the pair should be better ranked than the other one.

Perhaps the first Neural Network model for ranking is RankProp, originally proposed by [6]. RankProp is a pointwise approach that alternates between two phases of learning the desired real outputs by minimizing a MSE objective, and a modification of the desired values themselves to reflect the current ranking given by the net. Later on [5] proposed RankNet, a pairwise approach, that learns a preference function by minimizing a cross entropy cost over the pairs of relevant and irrelevant examples. SortNet proposed in [29, 30] also learns a preference function by minimizing a ranking loss over the pairs of examples that are selected iteratively with the overall aim of maximizing the quality of the ranking. The three approaches above consider the problem of Learning-to-Rank for IR and without learning an embedding.

Our work tackles the problem of learning user and item representations, based on neural language models, simultaneously with the learning of a preference function using Neural Networks which, to the best of our knowledge, is a new approach for Collaborative Filtering.

3 USER PREFERENCE AND EMBEDDING LEARNING WITH NEURAL NETS

This section is devoted to the presentation of $\texttt{RecNet}_{\alpha,\beta}$, the proposed Neural Network model, designed to learn user preference over items as well as their vector representations in an embedded space for collaborative filtering.

3.1 Problem Statement

We suppose that $\mathcal{U} = \{1, \ldots, N\}$ is the set of indexes over users, and $\mathcal{I} = \{1, \ldots, M\}$ the set of indexes over items. We denote by $\mathbf{U}$ and $\mathbf{V}$, the users and items latent features matrices respectively of size $N \times k$ and $M \times k$, where $\mathbf{U}_u$ and $\mathbf{V}_i$ correspond to the vector representations of user u and item i, and k is the dimension of the embedded space. Furthermore, to each user $u \in \mathcal{U}$ and item $i \in \mathcal{I}$ is associated a binary indicator vector, having all its characteristics equal to 0 except the one that indicates the position of u or i in their respective sets, which is equal to 1. Hence, the binary indicator vector of $u \in \mathcal{U}$ is

$$\mathbf{u}^\top = (0, \ldots, 0, 1, 0, \ldots, 0).$$

For any user $u \in \mathcal{U}$, we consider a pair of items $(i, i') \in \mathcal{I}^2$ over which u has a preference, symbolized by $\succ_u$, i.e. $i \succ_u i'$ implies that, user u prefers item i over item i'. From this preference relation a desired output $y_{i,u,i'} \in \{-1, +1\}$ is defined over each triplet $(i, u, i') \in \mathcal{I} \times \mathcal{U} \times \mathcal{I}$ as:

$$y_{i,u,i'} = \begin{cases} 1, & \text{if } i \succ_u i' \\ -1, & \text{if } i' \succ_u i \end{cases}. \tag{1}$$

The input of our model takes the binary indicator vectors of an item $i \in \mathcal{I}$, a user $u \in \mathcal{U}$ and a second item $i' \in \mathcal{I}$. The aim of learning is then to find an efficient prediction function

$$f : \{0,1\}^M \times \{0,1\}^N \times \{0,1\}^M \to [-1, +1],$$

that links an input $(\mathbf{i}, \mathbf{u}, \mathbf{i}') \in \{0,1\}^M \times \{0,1\}^N \times \{0,1\}^M$ to its corresponding desired output $y_{i,u,i'} \in \{-1, +1\}$. And also, to estimate an accurate users and items latent feature matrices $\mathbf{U}$ and $\mathbf{V}$. To achieve this goal, we propose the following double ranking objective function :

$$\mathcal{L}_r(f, \mathbf{U}, \mathbf{V}, \mathcal{S}) = \alpha \mathcal{L}_c(f, \mathcal{S}) + \beta \mathcal{L}_p(\mathbf{U}, \mathbf{V}, \mathcal{S}), \tag{2}$$

Where, $\mathcal{L}_c$ expresses the ability of the prediction function f to respect the relative ordering of items with respect to users preferences in the training set, $\mathcal{S}$. And, $\mathcal{L}_p$ expresses the ability of the dot-product to respect the relative ordering of items for users of the training set, in the embedded space induced by $\mathbf{U}$ and $\mathbf{V}$. In the remaining of the paper, $\mathcal{L}_c$ and $\mathcal{L}_p$ are referred to as the target loss and the embedding loss, respectively. The hyperparameters $\alpha \in [0, 1]$ and $\beta \in [0, 1]$ balance the compromise between prediction and expressiveness of the learned item and user representations.

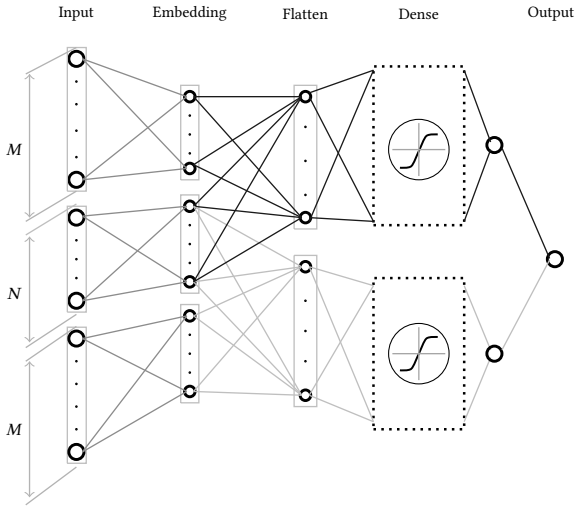


Figure 1: The architecture of $\texttt{RecNet}_{\alpha,\beta}$ composed of Embedding, Flatten and Dense hidden layers.

3.2 Model Architecture

$\texttt{RecNet}_{\alpha,\beta}$, depicted in Figure 1, is a feed-forward Neural Networks with sigmoid activation functions, where the input is the concatenation of binary indicator vector representations of an item i, a user u and another item i' and the output $y_{i,u,i'}$, is defined with respect to the relation $\succ_u$, (Equation 1). To reach the output from the input, there are three successive layers, namely *Embedding*, *Flatten* and *Dense* layers.

The Embedding layer is composed of three groups of units each fully connected to the binary indicator vectors of the items and the user, in the concatenated input vector. The Flatten layer is composed of two groups of units, each fully connected to the units in the Embedding layer and that corresponds to the dyadic representation of an item and a user. Each of these units are also fully connected

Algorithm 1 $\texttt{RecNet}_{\alpha,\beta}$**: Learning phase**

In:
T: maximal number of epochs
A set of users $\mathcal{U} = \{1, \ldots, N\}$
A set of items $\mathcal{I} = \{1, \ldots, M\}$
for $ep = 1, \ldots, T$
Randomly sample a mini-batch $\tilde{\mathcal{S}}_n$ of size n from the original user-item matrix
for all $((\mathbf{i}, \mathbf{u}, \mathbf{i}'), y_{i,u,i'}) \in \tilde{\mathcal{S}}_n$
Propagate $(\mathbf{i}, \mathbf{u}, \mathbf{i}')$ from the input to the output.
Retro-propagate the double ranking error $\mathcal{L}_r(f, \mathcal{U}, \mathcal{I}, \tilde{\mathcal{S}}_n)$, estimated over $\tilde{\mathcal{S}}_n$.
Out: Users and items latent feature matrices $\mathbf{U}, \mathbf{V}$ and the model weights.

Algorithm 2 $\texttt{RecNet}_{\alpha,\beta}$**: Test phase**

In:
A user $u \in \mathcal{U}$; A set of items $\mathcal{I} = \{1, \ldots, M\}$; A set containing the k preferred items in $\mathcal{I}$ by u; $\mathcal{G}_{u,k} \leftarrow \emptyset$;
The output of $\texttt{RecNet}_{\alpha,\beta}$ learned over a training set: f
Apply f to the first two items of $\mathcal{I}$ and, note the preferred one i^* and place it at the top of $\mathcal{G}_{u,k}$;
for $i = 3, \ldots, M$
if $f(i, u, i^*) > f(i^*, u, i)$ **then**
Add i to $\mathcal{G}_k$ at rank 1
else
$j \leftarrow 1$
while $j \leq k$ AND $f(\mathcal{G}_{u,k}(j), u, i) > f(i, u, \mathcal{G}_{u,k}(j))$
$j \leftarrow j + 1$
if $j \leq k$ **then**
Insert i in $\mathcal{G}_{u,k}$ at rank j
Out: $\mathcal{G}_k$;

to the units of a Dense layer composed of a succession of hidden layers.

The output of each of the dense layers reflects the relationship between the corresponding item and the user, presented at the input. From the input to the output layer, the architecture of $\texttt{RecNet}_{\alpha,\beta}$ is symmetric which ends up in the same mathematical expression of the outputs estimated by each of the dense layers. Let

$$g : \{0,1\}^N \times \{0,1\}^M \rightarrow [0,1],$$

be the composition function of layers from the input to the output of each of the dense layers. Hence, for an input $(\mathbf{i}, \mathbf{u}, \mathbf{i}')$, each of the dense layer predicts a score for the corresponding dyadic representation $(\mathbf{u}, \mathbf{i})$ or $(\mathbf{u}, \mathbf{i}')$ (i.e. $g(\mathbf{u}, \mathbf{i})$ or $g(\mathbf{u}, \mathbf{i}')$).

Finally, the prediction given by $\texttt{RecNet}_{\alpha,\beta}$ for an input $(\mathbf{i}, \mathbf{u}, \mathbf{i}')$ is defined upon g as:

$$f(\mathbf{i}, \mathbf{u}, \mathbf{i}') = g(\mathbf{u}, \mathbf{i}) - g(\mathbf{u}, \mathbf{i}'). \quad (3)$$

Thereby, $\texttt{RecNet}_{\alpha,\beta}$ allows to naturally define a pairwise ranking as a classification over the pairs, constituted by the dyadic representations of a user and each of preferred and less preferred items.

3.3 Learning and inference

For the sake of simplicity in implementation and computational efficiency, we considered the logistic surrogate ranking loss for the definitions of $\mathcal{L}_c$ and $\mathcal{L}_p$ estimated over mini-batches of size n. Each mini-batch

$$\tilde{\mathcal{S}}_n = \left\{((\mathbf{i}, \mathbf{u}, \mathbf{i}'), y_{i,u,i'}); (i, u, i') \in \mathcal{I} \times \mathcal{U} \times \mathcal{I}\right\},$$

is constituted by n triplets $(i, u, i') \in \mathcal{I} \times \mathcal{U} \times \mathcal{I}$ randomly sampled from the original user-item matrix and for which the preference relation $\succ_u$ holds. In this case the target loss $\mathcal{L}_c$ writes:

$$\mathcal{L}_c(f, \tilde{\mathcal{S}}_n) = \frac{1}{n} \sum_{((\mathbf{i},\mathbf{u},\mathbf{i}'), y_{i,u,i'}) \in \tilde{\mathcal{S}}_n} \log(1 + e^{y_{i,u,i'}(g(\mathbf{u},\mathbf{i}') - g(\mathbf{u},\mathbf{i}))}). \quad (4)$$

Similarly, the definition of the embedding loss $\mathcal{L}_p$ follows the goal of preserving the ordering induced by $\succ_u$ over the examples $(\mathbf{i}, \mathbf{u}, \mathbf{i}')$, $y_{i,u,i'}) \in \tilde{\mathcal{S}}_n$ in the learned embedded space by the dot product function defined in that space:

$$\mathcal{L}_p(\mathbf{U}, \mathbf{V}, \tilde{\mathcal{S}}_n) = \frac{1}{n} \sum_{((\mathbf{i},\mathbf{u},\mathbf{i}'), y_{i,u,i'}) \in \tilde{\mathcal{S}}_n} \log(1 + e^{y_{i,u,i'} \mathbf{U}_u^\top (\mathbf{V}_{i'} - \mathbf{V}_i)}). \quad (5)$$

In $\texttt{RecNet}_{\alpha,\beta}$, we used the back-propagation technique [4] to compute the error gradients for the weights over different mini-batches. The pseudo-code of the learning phase is shown in Algorithm 1. Beginning from a set of randomly chosen weights, the algorithm iterates until a maximum number of epochs T is reached.[2] At each epoch, a mini-batch $\tilde{\mathcal{S}}_n$ is randomly chosen from the original user-item matrix. The concatenated binary vectors of examples in $\tilde{\mathcal{S}}_n$ are then propagated throughout the network, and the double ranking error (Eq. 2) is retro-propagated using the descent gradient algorithm.

For the inference, shown in algorithm 2, a ranked list $\mathcal{G}_{u,k}$ of the $k \ll M$ most preferred items for each user in the test set is maintained while retrieving the set $\mathcal{I}$. Given the latent feature vectors of items and the user as well as the model weights; the two first items in $\mathcal{I}$ are placed in $\mathcal{G}_{u,k}$ in a way that the preferred one, i^*, is put at the top. The algorithm then retrieves the next item, $i \in \mathcal{I}$ by comparing its preference to i^*. This is simply carried out by comparing the model's output over the concatenated binary indicator vectors of $(\mathbf{i}^*, \mathbf{u}, \mathbf{i})$ and $(\mathbf{i}, \mathbf{u}, \mathbf{i}^*)$.

Hence, if $f(\mathbf{i}, \mathbf{u}, \mathbf{i}^*) > f(\mathbf{i}^*, \mathbf{u}, \mathbf{i})$, which from Equation 3 becomes equivalent to $g(\mathbf{u}, \mathbf{i}) > g(\mathbf{u}, \mathbf{i}^*)$, then i is predicted to be preferred over i^*; $i \succ_u i^*$and it is put at the first place instead of i^* in $\mathcal{G}_{u,k}$. Here we assume that the predicted preference relation $\succ_u$ is transitive, which then ensures that the predicted order in the list is respected. Otherwise, if i^* is predicted to be preferred over i, then i is compared to the second preferred item in the list, using the model' prediction as before, and so on. The new item, i, is inserted in $\mathcal{G}_{u,k}$ in the case if it is found to be preferred over another item in $\mathcal{G}_{u,k}$.

[2] Another stopping criteria, based on the relative errors of $\mathcal{L}_c$ between two epochs could be chosen as well.

By repeating the process until the end of $\mathcal{I}$, we obtain a ranked list of the k most preferred items for the user u. Algorithm 2 does not require an ordering of the whole set of items, as also in most cases we are just interested in the relevancy of the top ranked items for assessing the quality of a model. Further, its complexity is at most $O(k \times M)$ which is convenient in the case where $M \gg 1$. The merits of a similar algorithm have been discussed in [2] but, as pointed out above, the basic assumption for inserting a new item in the ranked list $\mathcal{G}_{u,k}$ is that the predicted preference relation induced by the model should be transitive, which may not hold in general.

In our experiments, we also tested a more conventional inference algorithm, which for a user u, consists in ordering the items in $\mathcal{I}$ with respect to the output given by the function g, and we did not find any substantial difference in all results, presented in the following sections, and obtained over different collections.

4 EXPERIMENTS

We conducted a number of experiments aimed at evaluating how the simultaneous learning of user and items representations, as well as the preferences of users over items can be efficiently handled with $\texttt{RecNet}_{\alpha,\beta}$ over two scenarios of implicit and explicit feedback. To this end, we considered six real-world benchmarks commonly used for collaborative filtering where the first three deal with explicit feedback, in the form of ratings, and the last three concern implicit feedback, in the form of clicks or binary data (Section 4.1). We validated our approach with respect to different effects that impact the model' behaviour and also by comparing it to competitive state-of-the-art approaches (Section 4.3).

4.1 Datasets

We report results obtained on three publicly available movie datasets, for the task of personalised top-N recommendation: MovieLens[3] 100K (ML-100K), MovieLens 1M (ML-1M) [13], Netflix, and three implicit datasets: we built the first two ones from the RecSys 2016[4] and the Outbrain[5] challenges, and the third one is a binarized version of MovieLens 20M (ML-20M). Table 1 provides basic statistics on these collections after preprocessing, as discussed below.

MovieLens and Netflix.
ML-100K, ML-1M and Netflix consist of user-movie ratings, on a scale of one to five, collected from a movie recommendation service and the Netflix company. This latter was released to support the Netlfix Prize competition[6]. ML-100K dataset gathers 100,000 ratings from 943 users on 1682 movies, ML-1M dataset consists of 1,000,000 ratings from 6040 users and 3900 movies and Netflix consist of 100 millions ratings from 480,000 users and 17,000 movies. For all three datasets, we only keep users who have rated at least five movies and remove users who gave the same rating for all movies. In addition, for Netflix, we take a subset of the original data and randomly sampled 20% of the users and 20% of the items.

[3]https://movielens.org/
[4]http://2016.recsyschallenge.com/
[5]https://www.kaggle.com/c/outbrain-click-prediction
[6]B. James and L. Stan, The Netflix Prize (2007).

RecSys 2016 clicks and Outbrain clicks.
We created two datasets out of the last RecSys 2016 Challenge and the ongoing Outbrain Challenge (Kaggle). For RecSys 2016, the initial task given by the challenge was to predict those job postings that a user will positively interact with (e.g. click, bookmark). For Outbrain, the task given by the challenge is to recommend content (e.g. news) that will most likely interest a user. For both datasets, we limited ourselves to the task of click prediction. In addition, we only keep users who clicked at least 5 times and did not click on at least 5 of the offers that were shown to them. For each user, thus, we keep at least 5 offers with a positive feedback (which they click) and at least 5 offers with a negative feedback (which they were shown and chose to ignore).

Binarized MovieLens 20 Million.
As for ML-100K and ML-1M, this data set is a collection of 20 millions ratings collected from more than 100,000 users over 11,000 of users. Following [21], we binarize the data by setting rating higher than three to one, and to zero otherwise. Similarly to other datasets, we only keep users who have rated at least five movies.

	# of users	# of items	# of interactions	Sparsity
ML-100K	943	1,682	100,000	93.7%
ML-1M	6040	3706	1,000,209	95.53%
Netflix	96,039	3,562	4,188,098	98.8%
RecSys 2016	106,398	641,075	16,362,222	99.97%
Outbrain	49,615	105,176	1,597,426	99.97%
ML-20M	137,768	26,740	19,972,149	99.46%

Table 1: Statistics of various collections used in our experiments after preprocessing.

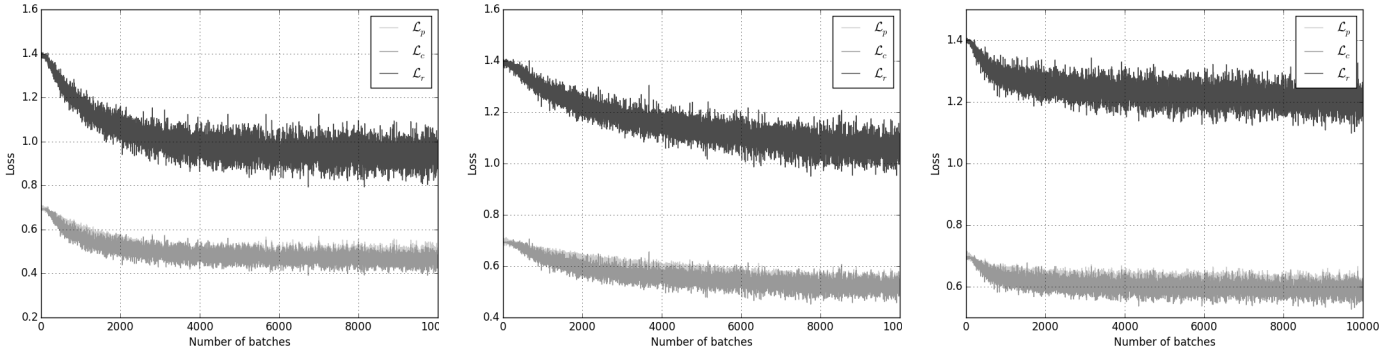
4.2 Experimental setup

To validate the double ranking approach described in the previous section, we tested the following seven methods.

The first two methods are the proposed model learned with only one of the ranking losses $\mathcal{L}_p$ (Eq. 5) or $\mathcal{L}_c$ (Eq. 4), and respectively correspond to the situations where ($\alpha = 0, \beta = 1$); $\texttt{RecNet}_{0,1}$ and ($\alpha = 1, \beta = 0$); $\texttt{RecNet}_{1,0}$. The goal here is to explore the impact of each of the ranking losses in (Eq. 2) and, in the overall performance of the model. The last configuration is obtained by varying the values of α and β in $[0.1, 1]$ and is referred to as $\texttt{RecNet}_{\alpha,\beta}$ in the following.

In order to see the effect of the embedding, we also trained the model by removing the corresponding layer (*Embedding* in Figure 1) and referred to as $\texttt{RecNet}_{E\setminus}$ in the following. Finally we compared, with the following three state-of-the-art methods that were exclusively developed for implicit or explicit feedback.

- BoostMF [7] is a collaborative filtering approach developed for explicit feedback, and it performs ranking by directly learning latent factors that are optimised toward the personalized top-K recommendations. This approach was found to outperform most of the other strong state-of-the-art ranking-oriented CF approaches, including ListRank



(a) ML-100K **(b) ML-1M** **(c) Netflix**

Figure 2: Evolution of the embedding loss, $\mathcal{L}_p$, the net loss, $\mathcal{L}_c$, and the double ranking loss $\mathcal{L}_r$ (Eq. 2) with respect to the number of epochs, for $\alpha = \beta = 1$ and the learning rate $\eta = 0.0001$ on ML-100K (a), ML-1M (b) and Netflic (c) collections. In all cases, $\mathcal{L}_p$ and, $\mathcal{L}_c$ start to overlap after only a very few number of batches.

[33] and CofiRank [37], as well as rating-oriented CF algorithms, including PMF [31], OrdRec [18] and UMR [34].

- BPR-MF [27] provides an optimization criterion based on implicit feedback; which is the maximum posterior estimator derived from a Bayesian analysis of the pairwise ranking problem, and proposes an algorithm based on Stochastic Gradient Descent to optimize it. The model can further be extended to the explicit feedback case.
- CoFactor [21], developed for implicit feedback, constraints the objective of matrix factorization to use jointly item representations with a factorized shifted positive pointwise mutual information matrix of item co-occurrence counts. The model was found to outperform WMF [14] also proposed for implicit feedback.

In order to compare the performance of these approaches, we first used the common truncated Normalized Discounted Cumulative Gain (NDCG@k) [15] in both implicit and explicit cases. This metric assumes that highly relevant items are more useful than negligible relevant ones, and that the lower the position of a selected item, the less useful it is for the user. DCG@k is defined as

$$\text{DCG@}k = \frac{\sum_{\ell=1}^{k}(2^{r_\ell} - 1)}{\log_2(1+k)},$$

where r_ℓ is the relevance judgment of the item of rank ℓ. Then, NDCG@k is the DCG@k normalized by an appropriate constant in order to lie in the interval $[0, 1]$.

Further, for the implicit case where the relevance judgments are binary, we also measured the Mean Average Precision (MAP@k) over all users in the test set. Where the Average Precision (AP@k) is defined over the precision, Pr, at rank ℓ.

$$\text{AP@}k = \frac{1}{k}\sum_{\ell=1}^{k} r_\ell Pr(\ell)$$

For the evaluation, we used the same protocol than in [3, 7] by considering three different settings: (1) for each user, we randomly selected 10 items for training, and the remaining items are used for the testing , then for (2) and (3) we randomly selected 20 and 30 items respectively, and proceed in the same manner. In order to ensure the computational feasibility of all metrics, users with less than 20, 30 and 60 rated items are removed in each of these settings. The performance of each method is estimated by averaging measure values obtained at each 10 repetitions over test items of all users. In the remainder of this section, these different settings are referred to as $S1$, $S2$ and $S3$ respectively.

Lastly, to train the `RecNet`. models we used Algorithm 1, and fixed the number of epochs to $T = 10,000$ and the size of the mini-batches to $n = 512$. Further, we used Adam [17] for the optimization of the double ranking loss, $\mathcal{L}_r$ and the learning rate η is chosen among the set $\{0.001, 0.0005, 0.0001\}$ using a validation set. For other parameters involved in Adam, i.e., the exponential decay rates for the moment estimates, we kept the default values ($\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$) and also used the L2 regularization with the values of the regularization term chosen among $\lambda \in \{0.0001, 0.001, 0.003\}$ using the same validation set. Moreover, for the computation of the score of the user-item pairs, we used a single hidden layer architecture with logistic activation functions and varied the number of hidden units in the set $\{32, \ldots, 256\}$.

We run all experiments on a cluster of five 32 core Intel Xeon @ 2.6Ghz CPU (with 20MB cache per core) systems with 256 Gig RAM running Debian GNU/Linux 8.6 (wheezy) operating system. All subsequently discussed components were implemented in Python3 using the TensorFlow library [1].[7]

[7]For research purpose we will make available all the codes implementing Algorithms 1 and 2 that we used in our experiments as well as all the preprocessed datasets.

4.3 Experimental Results

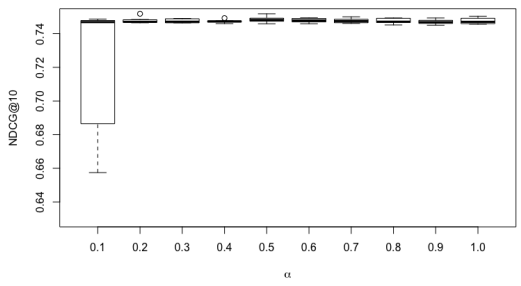
We start our evaluation by analyzing the learning ability of $\texttt{RecNet}_{\alpha,\beta}$. Figure 2 depicts the evolution of the embedding and the target losses, as well as the double ranking defined at the top of them with respect to the number of epochs on ML-100K, ML-1M and Netflix collections. As it can be observed, both losses, $\mathcal{L}_p$ and $\mathcal{L}_c$, gradually decrease as the epochs progress, and which consequently cause the decrease of the overall double ranking loss $\mathcal{L}_r$. This is an empirical evidence that both the objectives of, learning the representations of items and users and the preference of users over the pairs of items, measured by $\mathcal{L}_r$ can effectively be carried out by the proposed model.

4.3.1 The effect of the hyperparameters. Our approach requires a fine hyperparameters tuning. Indeed, besides the traditional hyperparameters involved in neural networks such as the learning rate or the number of hidden units, we also have to fix the values of α and β which balance the importance given to the embedding and the target losses. In our experiments, these hyperparameters were tuned over the same validation set; as all the other hyperparameters. In order to see this effect, in figure 3 we show the variation of NDCG@10 on ML-1M and Netflix datasets with respect to the values of $\alpha \in [0, 1]$ on the x-axis. For each value of α, the variations of NDCG@10 are shown in boxplots for values of $\beta \in [0, 1]$. On both figures, we note that when $\alpha = 0.1$ there is a huge gap in the performance of NDCG@10 and that for any values of β.

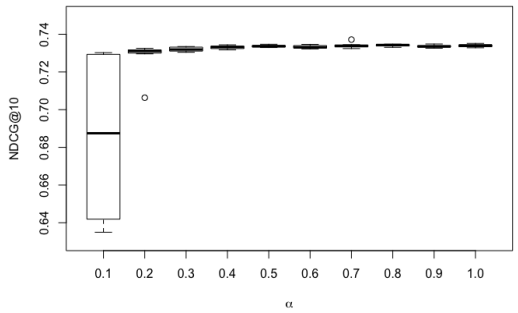
On the other hand, when the values of α exceeds a certain threshold (here $\alpha = 0.2$ in both cases), different values of β have less impact on the overall performance of the model. These results stress the importance of the embedding loss, $\mathcal{L}_p$, in the double ranking objective $\mathcal{L}_r$, and show the role of the representation learning in the proposed strategy.

4.3.2 The effect of the Embedded layer. To illustrate this role, we made experiments by completely removing the Embedded layer and learning the Neural-Networks without it ($\texttt{RecNet}_{\not{E}}$) using the same Algorithm 1. Figure 4 shows the histograms of NDCG@5 and NDCG@10 for $\texttt{RecNet}_{\not{E}}$ and $\texttt{RecNet}_{0,1}$ on Netflix for the three considered scenarios S1, S2 and S3. In order to have a fair comparison between both models, we fixed the value of α to 0, for $\texttt{RecNet}_{\alpha,\beta}$. In this way, the effect of the *Embedding* Layer becomes apparent as the information in the concatenated input vector will be first projected to the embedded space at the propagation phase of Algorithm 1, before being propagated through the Networks. Note that in the extreme case of $\alpha = 0$, there is no effect of the embedded loss on the updates of the weights between the input and the embedded layer. From these results, it comes out that the NDCG of $\texttt{RecNet}_{0,1}$ are 2% to 5% higher than those of $\texttt{RecNet}_{\not{E}}$ and that for different scenarios considered in this work. These results shed light on the importance of the embedded layer where the binary information is projected before its diffusion throughout the Network.

4.3.3 Learning with explicit feedback. We now present the results obtained for Boost-MF, BPR-MF, $\texttt{RecNet}_{1,0}$, $\texttt{RecNet}_{0,1}$, and $\texttt{RecNet}_{\alpha,\beta}$ over all explicit datasets and the three scenarios, in tables 2,3 and 4. The results of BPR-MF and BoostMF are directly reported from [7] as we rigorously followed their experimental setup in our work. From these results, there are several points to be made: on ML-100k and ML-1M, $\texttt{RecNet}_{\alpha,\beta}$ performs better (and in some cases substantially better) than Boost-MF and BPR-MF.

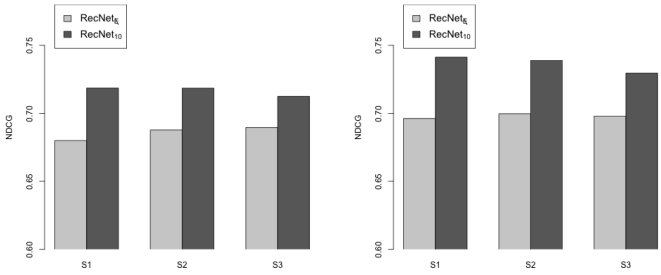


(a) ML-1M



(b) Netflix

Figure 3: Evolution of NDCG@10 obtained with $\texttt{RecNet}_{\alpha,\beta}$ for the values of (α;β) in [0,1] on ML-1M and Netflix collections.



(a) NDCG@5 **(b) NDCG@10**

Figure 4: Results obtained on Netflix in terms of NDCG@5 (left) and NDCG@10 (right) for $\texttt{RecNet}_{1,0}$ and for the Neural Networks without the embedding layer, $\texttt{RecNet}_{\not{E}}$.

However, we notice that compared to Boost-MF, the gap of performance is less important for NDCG@5 than for NDCG@10. This phenomena can be explained, as $\texttt{RecNet}_{\alpha,\beta}$ is based on pairwise ranking and, that it is learned using the logistic surrogate loss over the average number of disordered pairs. Hence, with $\texttt{RecNet}_{\alpha,\beta}$, the misoredering is less penalized for the top ranked items in a list, while Boost-MF is designed to correct the misorderings of the top-k ranked elements. The same observation holds on Netflix, with a slight advantage for Boost-MF this time. The main difference of Netflix and ML collections is the number of users that is 4 times more than in ML-1M. Our conjecture is that, the deduction of a preference relation over a predefined set of scores is not always valid, as depending on the humor (or people); two items may have two different scores without being preferred one over the other, and that even more on datasets with larger number of users like Netflix.

At this point, we can state the values of the hyperparameters of $\texttt{RecNet}_{\alpha,\beta}$ tuned using a validation set, on both ML-100K and ML-100M are; 84 hidden units, $\eta = 0.0001$ and $\lambda = 1e^{-3}$. On Netflix, we retain 128 hidden units with the same learning rate than previously and $\lambda = 0.001$. For α and β, we chose the best combination by varying those two in [0,1]. We retain the following combinations: (α, β)=$(0.4, 0.9)$, $(1, 1)$, $(1, 0.3)$ for ML-100K, ML-1M and Netflix, respectively.

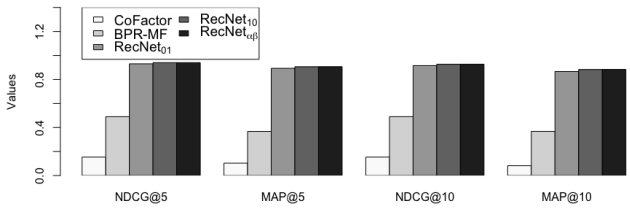
For the implicit case, for which the results are shown above we obtained that the combination $(1, 1)$ always provide the best results.

4.3.4 Learning with implicit feedback. We now compare $\texttt{RecNet}_{1,0}$, $\texttt{RecNet}_{0,1}$, and $\texttt{RecNet}_{\alpha,\beta}$ with BPR-MF and CoFactor [21], both originally developed to deal with implicit feedback. For the implementation of BPR-MF we used Mymedialite, a publicly available software [8]. For CoFactor, we ran the implementation provided by its authors[9].

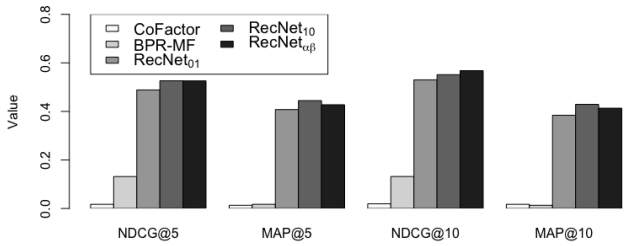
Figures 5 (a), (b) and (c), show the NDCG@5 (left) and NDCG@10 (right) obtained on respectively ML-20M, Outbrain, RecSys 2016 and for all the models except CoFactor on RecSys 2016. Indeed, for the construction of the context matrix that is involved in the learning embeddings, CoFactor uses timestamps to order items in a chronological way; and in RecSys 2016 we originally did not collect such timestamps. Further, we only report the results that correspond to the scenario S3, as for scenarios S1 and S2 the conclusions were the same.

We note that the results of CoFactor on ML-20M are consistent with those reported by the authors on their paper [21]. Furthermore, we observe that both NDCG of all three versions of RecNet. are higher than 0.8, 0.6 and 0.45 on respectively ML-20M, Outbrain and RecSys 2016 collections and that they outperform the two state-of-the-art models. Compared to the explicit feedback case discussed in the previous section, the good results obtained here, are likely due to a stronger preference relation induced from the implicit feedback contained in RecSys 2016 and Outbrain collections. Also, the binariztion of the ML-20M collection is with respect to a threshold, as discussed in Section 4.1, making that the preference relation induced from these data is more robust than the preference relation induced from the scores.

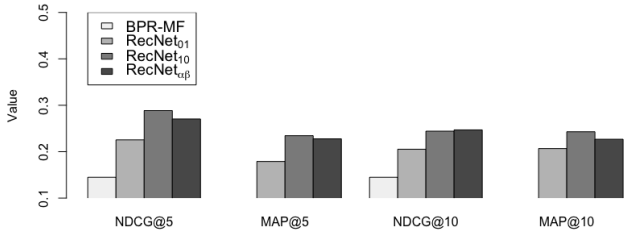
[8] http://mymedialite.net/download/index.html
[9] https://github.com/dawenl/cofactor



(a) ML-20M



(b) Outbrain



(c) RecSys 2016

Figure 5: Results obtained on RecSys 2016-Click, ML-20M and Outbrainin terms of NDCG@5 and NDCG@10 for BPR-MF, $\texttt{RecNet}_{0,1}$, $\texttt{RecNet}_{1,0}$ and $\texttt{RecNet}_{\alpha,\beta}$.

4.3.5 Complexity issues. In Table 5, we finally report the running times for the training and testing of $\texttt{RecNet}_{\alpha,\beta}$, in milliseconds, over the different collections used in our experiments. These times do not take into account the tuning of the hyperparameters that can be handled on a validation set separately, once and for all. As it can be observed, the longest time to train and to test $\texttt{RecNet}_{\alpha,\beta}$ is on RecSys 2016 collection, with approximately 6 and, a bit less than 4, minutes for respectively training and testing.

	$S1$		$S2$		$S3$	
	NDCG@5	NDCG@10	NDCG@5	NDCG@10	NDCG@5	NDCG@10
BPR-MF	0.5558 ± .002	0.5942 ± .003	0.5872 ± .002	0.6098 ± .004	0.6292 ± .001	0.6309 ± .002
BoostMF	0.6722 ± .008	0.7034 ± .007	0.6921 ± .005	0.7019 ± .004	**0.7117 ± .004**	0.7135 ± .004
$\texttt{RecNet}_{0,1}$	0.6845 ± .003	0.6858 ± .004	0.6883 ± .004	0.7006 ± .003	0.6949 ± .003	0.6919 ± .002
$\texttt{RecNet}_{1,0}$	0.6837 ± .002	0.7012 ± .003	0.6860 ± .003	0.6928 ± .004	0.6913 ± .004	0.6979 ± .003
$\texttt{RecNet}_{\alpha,\beta}$	**0.6893 ± .001**	**0.7143 ± .002**	**0.7012 ± .002**	**0.7197 ± .001**	0.7034 ± .001	**0.7195 ± .002**

Table 2: Mean (± standard deviation) of NDCG@5 and NDCG@10, obtained for all compared methods, over 10 runs with each settings on ML-100K. $\texttt{RecNet}_{0,1}$ and $\texttt{RecNet}_{1,0}$ correspond to the situation ($\alpha = 0, \beta = 1$) and ($\alpha = 1, \beta = 0$), respectively. $\texttt{RecNet}_{\alpha,\beta}$ corresponds to the best results with (α, β) that vary between 0.1 and 1.

	$S1$		$S2$		$S3$	
	NDCG@5	NDCG@10	NDCG@5	NDCG@10	NDCG@5	NDCG@10
BPR-MF	0.6734 ± .007	0.6873 ± .006	0.6747 ± .005	0.6790 ± .006	0.6711 ± .007	0.6769 ± .006
BoostMF	**0.7433 ± .007**	0.7389 ± .007	0.7475 ± .005	0.7480 ± .004	0.7528 ± .004	0.7515 ± .004
$\texttt{RecNet}_{0,1}$	0.6760 ± .005	0.6819 ± .008	0.6916 ± .008	0.7147 ± .005	0.7397 ± .005	0.7167 ± .004
$\texttt{RecNet}_{1,0}$	0.7371 ± .003	0.7462 ± .002	**0.7494 ± .004**	0.7511 ± .002	0.7557 ± .002	0.7537 ± .003
$\texttt{RecNet}_{\alpha,\beta}$	0.7366 ± .004	**0.7504 ± .004**	0.7485 ± .002	**0.7567 ± .003**	**0.7589 ± .001**	**0.7550 ± .001**

Table 3: Mean (± standard deviation) of NDCG@5 and NDCG@10, obtained for all compared methods, over 10 runs with each settings on ML-1M. $\texttt{RecNet}_{0,1}$ and $\texttt{RecNet}_{1,0}$ correspond to the situation ($\alpha = 0, \beta = 1$) and ($\alpha = 1, \beta = 0$), respectively. $\texttt{RecNet}_{\alpha,\beta}$ corresponds to the best results with (α, β) that vary between 0.1 and 1.

	$S1$		$S2$		$S3$	
	NDCG@5	NDCG@10	NDCG@5	NDCG@10	NDCG@5	NDCG@10
BPR-MF	0.5724 ± .003	0.6005 ± .005	0.5725 ± .004	0.5982 ± .005	0.5783 ± .003	0.6095 ± .003
BoostMF	**0.7216 ± .006**	0.7364 ± .002	**0.7352 ± .004**	**0.7398 ± .006**	**0.7317 ± .003**	**0.7383 ± .002**
$\texttt{RecNet}_{0,1}$	0.6990 ± .005	0.6994 ± .002	0.6631 ± .007	0.6743 ± .004	0.6719 ± .005	0.6904 ± .003
$\texttt{RecNet}_{1,0}$	0.7186 ± .004	**0.7413 ± .003**	0.7185 ± .002	0.7389 ± .002	0.7125 ± .005	0.7286 ± .003
$\texttt{RecNet}_{\alpha,\beta}$	0.7098 ± .002	0.7372 ± .001	0.7186 ± .002	0.7352 ± .001	0.7184 ± .002	0.7381 ± .002

Table 4: Mean (± standard deviation) of NDCG@5 and NDCG@10, obtained for all compared methods, over 10 runs with each settings on Netflix. $\texttt{RecNet}_{0,1}$ and $\texttt{RecNet}_{1,0}$ correspond to the situation ($\alpha = 0, \beta = 1$) and ($\alpha = 1, \beta = 0$), respectively. $\texttt{RecNet}_{\alpha,\beta}$ corresponds to the best results with (α, β) that vary between 0.1 and 1.

5 CONCLUSION AND DISCUSSION

We presented $\texttt{RecNet}_{\alpha,\beta}$, a new Neural-Network based approach for Collaborative Filtering, where both the user and item representations in an embedded space and the prediction function translating the users preference over pairs of items are learned simultaneously. The learning phase is guided using a double ranking objective that measures the ranking ability of the prediction function as well as the expressiveness of the learned embedded space, where the preference of users over items is respected by the dot product function defined over that space. The training of $\texttt{RecNet}_{\alpha,\beta}$ is carried out using the back-propagation algorithm on mini-batches defined over a user-item matrix containing, either implicit information in the form of subsets of preferred and no-preferred items for each user, or explicit information in the form of assigned scores to items. The learnability of the model over both prediction and representation problems show their interconnection and also that the proposed double ranking objective allows to conjugate well between them.

We evaluated and validated our approach using six implicit and explicit collections proposed for Collaborative Filtering. We assessed through extensive experiments the validity of our proposed approach. More precisely, we demonstrated that the definition of RecNet linking a triplet constituted by a user and two items, with an associated output, based on the preference of the user over those items, makes the model to naturally tackle both explicit and implicit feedback. For the latter, we noticed an important gap between the performance of the RecNet and strong state-of-the-art baselines proposed for this task.

For future work, we would like to extend RecNet in order to take into account additional contextual information regarding users and/or items. More specifically, we are interested in the integration of data of different natures, such as text or demographic information. We believe that these information can be taken into account without much effort and by doing so, it is possible to improve the performance of our approach and tackle the problem of providing recommendation for new users/items at the same time, also known as the cold-start problem. The second important extension will be

	training time	testing time
ML-100K	52,458	48
ML-1M	56,231	37,821
Netflix	69,836	32,123
RecSys 2016	361,327	201,438
Outbrain	153,211	41,721
ML-20M	207,679	104,313

Table 5: Complexity of our approach on all benchmarks data for the S3 setting. Times are expressed in milliseconds and were computed for the all training phase (10000 batches) and the all prediction phase.

the development of an on-line version of the proposed algorithm in order to make the approach suitable for real-time applications and on-line advertising.

ACKNOWLEDGMENTS

The work of Oleh Horodnitskii and Yury Maximov at Skolkovo Institute of Science and Technology is supported in part by the grant of the President of Russia for young PhD MK-9662.2016.9 and by the RFBR grant 15-07-09121a. The work at LANL was carried out under the auspices of the National Nuclear Security Administration of the U.S. Department of Energy under Contract No. DE-AC52-06NA25396.

REFERENCES

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. (2015). http://tensorflow.org/ Software available from tensorflow.org.

[2] Nir Ailon and Mehryar Mohri. 2008. An efficient reduction of ranking to classification. In *Conference On Learning Theory, (COLT)*. 87–98.

[3] Suhrid Balakrishnan and Sumit Chopra. 2012. Collaborative Ranking. In *Proceedings of the Fifth ACM International Conference on Web Search and Data Mining (WSDM '12)*. ACM, New York, NY, USA, 143–152.

[4] Léon Bottou. 2012. Stochastic Gradient Descent Tricks. In *Neural Networks: Tricks of the Trade - Second Edition*. 421–436.

[5] Christopher J. C. Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Gregory N. Hullender. 2005. Learning to rank using gradient descent. In *Machine Learning, Proceedings of the Twenty-Second International Conference (ICML 2005)*. 89–96.

[6] Rich Caruana, Shumeet Baluja, and Tom Mitchell. 1995. Using the Future to "Sort out" the Present: Rankprop and Multitask Learning for Medical Risk Evaluation. In *Proceedings of the 8th International Conference on Neural Information Processing Systems (NIPS'95)*. 959–965.

[7] Nipa Chowdhury, Xiongcai Cai, and Cheng Luo. 2015. *BoostMF: Boosted Matrix Factorisation for Collaborative Ranking*. Springer International Publishing, Cham, 3–18.

[8] William W. Cohen, Robert E. Schapire, and Yoram Singer. 1999. Learning to order things. *Journal of Artificial Intelligence Research* 10, 1 (1999).

[9] Koby Crammer and Yoram Singer. 2001. Pranking with Ranking. In *Advances in Neural Information Processing Systems (NIPS 14)*. MIT Press.

[10] Yoav Freund, Raj Iyer, Robert E. Schapire, and Yoram Singer. 2003. An efficient boosting algorithm for combining preferences. *Journal of Machine Learning Research* (2003).

[11] Mihajlo Grbovic, Vladan Radosavljevic, Nemanja Djuric, Narayan Bhamidipati, Jaikit Savla, Varun Bhagwan, and Doug Sharp. 2015. E-commerce in Your Inbox: Product Recommendations at Scale. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Sydney, NSW, Australia, August 10-13, 2015*. 1809–1818.

[12] E. Guàrdia-Sebaoun, V. Guigue, and P. Gallinari. 2015. Latent Trajectory Modeling: A Light and Efficient Way to Introduce Time in Recommender Systems. In *Proceedings of the 9th ACM Conference on Recommender Systems (RecSys '15)*. 281–284.

[13] F. Maxwell Harper and Joseph A. Konstan. 2015. The MovieLens Datasets: History and Context. *ACM Trans. Interact. Intell. Syst.* 5, 4, Article 19 (Dec. 2015), 19 pages.

[14] Yifan Hu, Yehuda Koren, and Chris Volinsky. 2008. Collaborative Filtering for Implicit Feedback Datasets. In *Proceedings of the 2008 Eighth IEEE International Conference on Data Mining (ICDM '08)*. 263–272.

[15] Kalervo Järvelin and Jaana Kekäläinen. 2002. Cumulated Gain-based Evaluation of IR Techniques. *ACM Trans. Inf. Syst.* 20, 4 (Oct. 2002), 422–446.

[16] Thorsten Joachims. 2002. Optimizing search engines using clickthrough data. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 133–142.

[17] Diederik P. Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. *CoRR* abs/1412.6980 (2014).

[18] Yehuda Koren and Joe Sill. 2011. OrdRec: An Ordinal Model for Predicting Personalized Item Rating Distributions. In *Proceedings of the Fifth ACM Conference on Recommender Systems (RecSys '11)*. 117–124.

[19] O. Levy and Y. Goldberg. 2014. Neural Word Embedding as Implicit Matrix Factorization. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*. 2177–2185.

[20] Ping Li, Christopher Burges, and Qiang Wu. 2008. McRank: Learning to Rank Using Multiple Classification and Gradient Boosting. In *Advances in Neural Information Processing Systems (NIPS 20)*.

[21] D. Liang, J. Altosaar, L. Charlin, and D. M. Blei. 2016. Factorization Meets the Item Embedding: Regularizing Matrix Factorization with Item Co-occurrence. In *Proceedings of the 10th ACM Conference on Recommender Systems (RecSys '16)*. 59–66.

[22] Tie-Yan Liu. 2009. Learning to Rank for Information Retrieval. *Foundation Trends in Information Retrieval* 3, 3 (2009).

[23] Pasquale Lops, Marco de Gemmis, and Giovanni Semeraro. 2011. Content-based Recommender Systems: State of the Art and Trends. In *Recommender Systems Handbook*, Francesco Ricci, Lior Rokach, Bracha Shapira, and Paul B. Kantor (Eds.). Springer, 73–105.

[24] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient Estimation of Word Representations in Vector Space. *CoRR* abs/1301.3781 (2013).

[25] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed Representations of Words and Phrases and their Compositionality. In *Advances in Neural Information Processing Systems 26*. 3111–3119.

[26] Michael J. Pazzani and Daniel Billsus. 2007. The Adaptive Web. Springer-Verlag, Berlin, Heidelberg, Chapter Content-based Recommendation Systems, 325–341.

[27] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2009. BPR: Bayesian Personalized Ranking from Implicit Feedback. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence (UAI '09)*. AUAI Press, Arlington, Virginia, United States, 452–461.

[28] Francesco Ricci, Lior Rokach, Bracha Shapira, and Paul B. Kantor. 2010. *Recommender Systems Handbook* (1st ed.). Springer-Verlag New York, Inc., New York, NY, USA.

[29] Leonardo Rigutini, Tiziano Papini, Marco Maggini, and Monica Bianchini. 2008. A Neural Network Approach for Learning Object Ranking. In *Artificial Neural Networks - ICANN 2008, 18th International Conference, Prague, Czech Republic, September 3-6, 2008, Proceedings, Part II*. 899–908.

[30] Leonardo Rigutini, Tiziano Papini, Marco Maggini, and Franco Scarselli. 2011. SortNet: Learning to Rank by a Neural Preference Function. *IEEE Trans. Neural Networks* 22, 9 (2011), 1368–1380.

[31] Ruslan Salakhutdinov and Andriy Mnih. 2007. Probabilistic Matrix Factorization. In *Advances in Neural Information Processing Systems 20, Proceedings of the Twenty-First Annual Conference on Neural Information Processing Systems*. 1257–1264.

[32] Noam Shazeer, Ryan Doherty, Colin Evans, and Chris Waterson. 2016. Swivel: Improving embeddings by noticing what's missing. *arXiv preprint arXiv:1602.02215* (2016).

[33] Yue Shi, Martha Larson, and Alan Hanjalic. 2010. List-wise learning to rank with matrix factorization for collaborative filtering. In *Proceedings of the fourth ACM conference on Recommender systems (RecSys '10)*. ACM, 269–272.

[34] Yue Shi, Martha Larson, and Alan Hanjalic. 2013. Unifying Rating-oriented and Ranking-oriented Collaborative Filtering for Improved Recommendation. *Inf. Sci.* 229 (April 2013), 29–39.

[35] Flavian Vasile, 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, Boston, MA, USA, September 15-19, 2016*. 225–232.

[36] Hao Wang, Naiyan Wang, and Dit-Yan Yeung. 2015. Collaborative Deep Learning for Recommender Systems. In *Proceedings of the 21th ACM SIGKDD International*

Conference on Knowledge Discovery and Data Mining, Sydney, NSW, Australia, August 10-13, 2015. 1235–1244.

[37] Markus Weimer, Alexandros Karatzoglou, Quoc Viet Le, and Alex Smola. 2007. COFIRANK Maximum Margin Matrix Factorization for Collaborative Ranking. In *Proceedings of the 20th International Conference on Neural Information Processing Systems (NIPS'07)*. 1593–1600.

[38] R. W. White, J. M. Jose, and I. Ruthven. 2001. Comparing explicit and implicit feedback techniques for web retrieval: TREC-10 interactive track report. In *Proceedings of the Tenth Text Retrieval Conference (TREC-10)*, E. Voorhees and D. Harman (Eds.). NIST, 534–538.

[39] Jun Xu and Hang Li. 2007. AdaRank: a boosting algorithm for information retrieval. In *SIGIR 2007: Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. 391–398.

[40] Jun Xu, Tie-Yan Liu, Min Lu, Hang Li, and Wei-Ying Ma. 2008. Directly optimizing evaluation measures in learning to rank. In *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2008, Singapore, July 20-24, 2008*. 107–114.