Hybrid Collaborative Filtering with Neural Networks

Florian Strub, Jérémie Mary, Romaric Gaudel

Hybrid Collaborative Filtering with Neural Networks

Florian Strub FLORIAN.STRUB@INRIA.FR
Univ of Lille, CRIStAL, Inria, SequeL Team, 40 av. du Halley, 59650 Villeneuve d'Ascq, FRANCE

Jérémie Mary JEREMIE.MARY@INRIA.FR
Univ of Lille, CRIStAL, Inria, SequeL Team, 40 av. du Halley, 59650 Villeneuve d'Ascq, FRANCE

Romaric Gaudel ROMARIC.GAUDEL@INRIA.FR
Univ of Lille, CRIStAL, Inria, SequeL Team, 40 av. du Halley, 59650 Villeneuve d'Ascq, FRANCE

Abstract

Collaborative Filtering aims at exploiting the feedback of users to provide personalised recommendations. Such algorithms look for latent variables in a large sparse matrix of ratings. They can be enhanced by adding side information to tackle the well-known cold start problem. While Neural Networks have tremendous success in image and speech recognition, they have received less attention in Collaborative Filtering. This is all the more surprising that Neural Networks are able to discover latent variables in large and heterogeneous datasets. In this paper, we introduce a Collaborative Filtering Neural network architecture aka CFN which computes a non-linear Matrix Factorization from sparse rating inputs and side information. We show experimentally on the MovieLens and Douban dataset that CFN outperforms the state of the art and benefits from side information. We provide an implementation of the algorithm as a reusable plugin for Torch, a popular Neural Network framework.

1. Introduction

Recommendation systems advise users on which items (movies, musics, books etc.) they are more likely to be interested in. A good recommendation system may dramatically increase the amount of sales of a firm or retain customers. For instance, 80% of movies watched on Netflix come from the recommender system of the company (Gomez-Uribe & Hunt, 2015). One efficient way to design such algorithm is to predict how a user would rate a given item. Two key methods co-exist to tackle this issue: *Content-Based Filtering* and *Collaborative Filtering* (CF).

Preliminary work. Under review by the under review by the International Conference on Machine Learning (ICML),

Content-Based Filtering explicitly uses the user/item knowledge to estimate a new rating. For instance, user information can be the age, gender or graph of friends etc. Item information can be the movie genre, a short description or the tags. CF uses the ratings history of users and items. The feedback of *one* user on *some* items is combined with the feedback of *all* other users on *all* items to predict a new rating. For instance, if someone rated a few books, Collaborative Filtering aims at estimating the ratings he would have given to thousands of other books by using the ratings of all the other readers. CF is often preferred to Content-Based Filtering because it wins the agnostic vs. studied contest: CF only relies on the ratings of the users while Content-Based Filtering requires advanced engineering on items to perform well (Lops et al., 2011).

The most successful approach in CF is to retrieve potential latent factors from the sparse matrix of ratings. Book latent factors are likely to encapsulate the book genre (spy novel, fantasy, etc.) or some writing styles. Common latent factor techniques compute a low-rank rating matrix by applying Singular Value Decomposition through gradient descent (Koren et al., 2009) or Regularized Alternative Least Square algorithm (Zhou et al., 2008). However, these methods are linear and cannot catch subtle factors. Newer algorithms were explored to face those constraints such as Non Linear Probabilistic Matrix Factorization (Lawrence & Urtasun, 2009), Factorization Machines (Rendle, 2010) or Local Low Rank Matrix Approximation (Lee et al., 2013).

Another limitation of CF is known as the *cold start* problem: how to recommend an item to a user when no rating exists for neither the user nor the item ? To overcome this issue, one idea is to build a hybrid model mixing CF and Content Based Filtering where side information is integrated into the training process. The goal is to supplant the lack of ratings through side information. A successful approach (Adams & Murray, 2010; Porteous & A. U. Asuncion, 2010) extends the Bayesian Probabilistic Matrix Factorization Framework (Salakhutdinov & Mnih, 2008) to in-

tegrate side information. However, recent algorithms outperform them in the general case (Lee et al., 2012).

In this paper we introduce a CF approach based on Stacked Denoising Autoencoders (Vincent et al., 2010) which tackles both challenges: learning a non-linear representation of users and items, and alleviating the cold start problem by integrating side information. Compared to previous attempts in that direction (Salakhutdinov et al., 2007; Sedhain et al., 2015; Strub & Mary, 2015; Dziugaite & Roy, 2015), our framework integrates the sparse matrix of ratings and side information in a unique Network. This joint model leads to a scalable and robust approach which beats state-of-the-art results in CF. Reusable source code is provided in Lua/Torch to reproduce the results. Last but not least, we show that CF approaches based on Matrix Factorization have a strong link with our approach.

The paper is organized as follows. First, Sec. 2 summarizes the state-of-the-art in CF and Neural Networks. Then, our model is described in Sec. 3 and 4 and its relation with Matrix Factorization is characterized in Sec. 3.2. Finally, experimental results are given and discussed in Sec. 5 and Sec. 6 discusses algorithmic aspects.

2. Preliminaries

2.1. Denoising Autoencoders

The proposed approach builds upon Autoencoders which are feed-forward Neural Networks popularized by Kramer (1991). They are unsupervised Networks where the output of the Network aims at reconstructing the initial input. The Network is constrained to use narrow hidden layers, forcing a dimensionality reduction on the data. The Network is trained by back-propagating the squared error loss on the reconstruction. Such Networks are divided into two parts:

- the encoder : $f(\mathbf{x}) = \sigma(\mathbf{W_1 x} + \mathbf{b_1})$,
- the decoder : $g(\mathbf{y}) = \sigma(\mathbf{W_2 y} + \mathbf{b_2})$,

with $\mathbf{x} \in \mathbb{R}^N$ the input, $\mathbf{y} \in \mathbb{R}^K$ the output, K the size of the Autoencoder's bottleneck ($K << N$), $\mathbf{W_1} \in \mathbb{R}^{N \times K}$ and $\mathbf{W_2} \in \mathbb{R}^{K \times N}$ the weight matrices, $\mathbf{b_1} \in \mathbb{R}^K$ and $\mathbf{b_2} \in \mathbb{R}^N$ the bias vectors, and $\sigma(.)$ a non-linear transfer function.

Recent work in Deep Learning advocates to stack pre-trained encoders to initialize Deep Neural Networks (Glorot & Bengio, 2010). This process enables the lowest layers of the Network to find low-dimensional representations. It experimentally increases the quality of the whole Network. Yet, classic Autoencoders often degenerate into identity Networks and they fail to learn the latent relationship between data. (Vincent et al., 2010) tackle this issue by corrupting inputs, pushing the Network to denoise the final outputs. One method is to add Gaussian noise on a random fraction ν of the input. Another method is to mask a random fraction ν of the input by replacing them with zero. In this case, the Denoising AutoEncoder (DAE) loss function is modified to emphasize the denoising aspect of the Network. The loss is based on two main hyperparameters α, β. They balance whether the Network would focus on denoising the input (α) or reconstructing the input (β):

$$L_{2,\alpha,\beta}(\mathbf{x}, \tilde{\mathbf{x}}) = \alpha \left(\sum_{j \in \mathcal{C}(\tilde{\mathbf{x}})} [nn(\tilde{\mathbf{x}})_j - x_j]^2 \right) + \beta \left(\sum_{j \notin \mathcal{C}(\tilde{\mathbf{x}})} [nn(\tilde{\mathbf{x}})_j - x_j]^2 \right),$$

where $\tilde{\mathbf{x}}$ is the corrupted input $\mathbf{x} \in \mathbb{R}^N$, $\mathcal{C}$ are the indices of the corrupted elements of $\mathbf{x}$, $nn(\mathbf{x})_j$ is the j^{th} output of the Network.

2.2. Matrix Factorization

One of the most successful approach of Collaborative Filtering is Matrix Factorization (Koren et al., 2009). This method retrieves latent factors from the ratings of items made by the users. The underlying idea is that key features are hidden in the ratings themselves. Given N users and M items, the rating r_{ij} is the rating given by the i^{th} user for the j^{th} item. It entails a sparse matrix of ratings $\mathbf{R} \in \mathbb{R}^{N \times M}$. In Collaborative Filtering, sparsity is originally produced by missing values rather than zero values. The goal of Matrix Factorization is to find a K low rank matrix $\hat{\mathbf{R}} \in \mathbb{R}^{N \times M}$ where $\hat{\mathbf{R}} = \mathbf{U}\mathbf{V}^T$ with $\mathbf{U} \in \mathbb{R}^{N \times K}$ and $\mathbf{V} \in \mathbb{R}^{M \times K}$ are two matrices of rank K encoding a dense representation of the users/items with

$$\underset{\mathbf{U},\mathbf{V}}{\arg\min} \sum_{(i,j) \in \mathcal{K}(\mathbf{R})} (r_{ij} - \bar{\mathbf{u}}_i^T \bar{\mathbf{v}}_j)^2 + \lambda(\|\bar{\mathbf{u}}_i\|_{Fro}^2 + \|\bar{\mathbf{v}}_j\|_{Fro}^2),$$

where $\mathcal{K}(\mathbf{R})$ is the set of indices of known ratings, $\bar{\mathbf{u}}_i$, $\bar{\mathbf{v}}_j$ are the corresponding line vectors of $\mathbf{U}$,$\mathbf{V}$ and $\|\bar{\mathbf{u}}_i\|_{Fro}$ is the Frobenius norm.

2.3. Related Work

Neural Networks have attracted little attention in the CF community. In a preliminary work, (Salakhutdinov et al., 2007) tackled the Netflix challenge using Restricted Boltzmann Machines but little published work had follow (Truyen et al., 2009). While Deep Learning has tremendous success in image and speech recognition (LeCun et al., 2015), sparse data has received less attention and remains a challenging problem for Neural Networks.

Nevertheless, Neural Networks are able to discover non-linear latent variables with heterogeneous data (LeCun et al., 2015) which makes them a promising tool for CF.

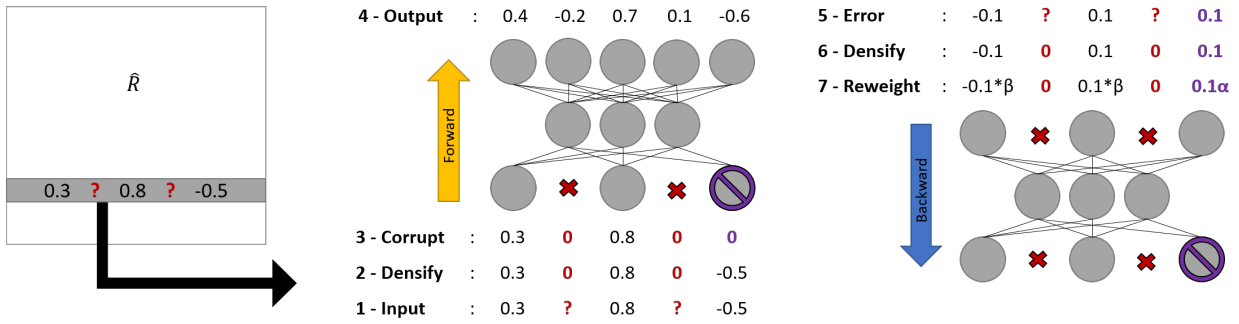

Figure 1. Feed Forward/Backward process for sparse Autoencoders. The sparse input is extracted from the matrix of ratings, unknown values are turned to zero, input is corrupted by masking additional ratings and a dense estimate is finally obtained. Before backpropagation, unknown ratings are turned to zero error, predicted errors are reweighed by α and reconstruction errors are reweighed by β.

(Sedhain et al., 2015; Strub & Mary, 2015; Dziugaite & Roy, 2015) directly train Autoencoders to provide the best predicted ratings. Those methods report excellent results in the general case. However, the cold start initialization problem is ignored. For instance, AutoRec (Sedhain et al., 2015) replaces unpredictable ratings by an arbitrary selected score. In our case, we apply a training loss designed for *sparse* rating inputs and we integrate side information to lessen the cold start effect.

Other contributions deal with this cold start problem by using Neural Networks properties for Content-Based Filtering: Neural Networks are first trained to learn a feature representation from the item which is then processed obtain a CF approach such as Probabilistic Matrix Factorization (Mnih & Salakhutdinov, 2007) to provide the final rating. For instance, (Glorot et al., 2011; Wang et al., 2014a) respectively auto-encode bag-of-words from restaurant reviews and movie plots, (Li et al., 2015) auto-encode heterogeneous side information from users and items. Finally, (den Oord et al., 2013; Wang et al., 2014b) use Convolutional Networks on music samples. In our case, side information and ratings are used together without any unsupervised pretreatment.

2.4. Notation

In the rest of the paper, we will use the following notations:

- $\mathbf{u_i}$, $\mathbf{v_j}$ are the sparse lines/columns of $\mathbf{R}$
- $\tilde{\mathbf{u}}_i$, $\tilde{\mathbf{v}}_\mathbf{j}$ are corrupted versions of $\mathbf{u_i}$, $\mathbf{v_j}$
- $\hat{\mathbf{u}}_i$, $\hat{\mathbf{v}}_j$ are dense estimates of $\hat{\mathbf{R}}$
- $\bar{\mathbf{u}}_i$, $\bar{\mathbf{v}}_j$ are dense low rank representations of $\mathbf{u_i}$, $\mathbf{v_j}$.

3. Autoencoders for Collaborative Filtering

User preferences are encoded as a sparse matrix of ratings $\mathbf{R}$. A user is represented by a sparse line $\mathbf{u}_i \in \mathbb{R}^N$ and an item is represented by a sparse column $\mathbf{v}_j \in \mathbb{R}^M$. The Collaborative Filtering objective can be formulated as: *turn the sparse vectors* $\mathbf{u}_i$/$\mathbf{v}_j$*, into dense vectors* $\hat{\mathbf{u}}_i$/$\hat{\mathbf{v}}_j$.

We propose to perform this conversion with Autoencoders. To do so, we need to define two types of Autoencoders:

- U-CFN is defined as $nn(\mathbf{u_i}) = \hat{\mathbf{u}}_i$,
- V-CFN is defined as $nn(\mathbf{v_j}) = \hat{\mathbf{v}}_j$.

The encoding part of these Autoencoder aims at building a low-rank dense representation of the sparse input of ratings. The decoding part aims at predicting a dense vector of ratings from the low-rank dense representation of the encoder. This new approach differs from classic Autoencoders which only aim at reconstructing/denoising the input. As we will see later, the training loss will then differ from the evaluation one.

3.1. Sparse Inputs

There is no standard approach for using sparse vectors as inputs of Neural Networks. Most of the papers dealing with sparse inputs get around by pre-computing an estimate of the missing values (Tresp et al., 1994; Bishop, 1995). In our case, we want the Autoencoder to handle this prediction issue by itself. Such problems have already been studied in industry (Miranda et al., 2012) where 5% of the values are missing. However in Collaborative Filtering we often face datasets with more than 95% missing values. Furthermore, missing values are *not* known during *training* in Collaborative Filtering which makes the task even more difficult.

Our approach includes three ingredients to handle the training of sparse Autoencoders:

- inhibit the edges of the input layers by zeroing out values in the input,
- inhibit the edges of the output layers by zeroing out back-propagated values,
- use a denoising loss to emphasize rating prediction over rating reconstruction.

One way to inhibit the input edges is to turn missing values to zero. To keep the Autoencoder from always returning zero, we also use an empirical loss that disregards the loss of unknown values. No error is back-propagated for missing values. Therefore, the error is back-propagated for actual zero values while it is discarded for missing values. In other words, missing values do not bring information to the Network. This operation is equivalent to removing the neurons with missing values described in (Salakhutdinov et al., 2007; Sedhain et al., 2015). However, Our method has important computational advantages because only one Neural Networks is trained whereas other techniques has to share the weights among thousands of Networks.

Finally, we take advantage of the masking noise from the Denoising AutoEncoders (DAE) empirical loss. By simulating missing values in the training process, Autoencoders are trained to predict them. In Collaborative Filtering, this prediction aspect is actually the final target. Thus, emphasizing the prediction criterion turns the classic unsupervised training of Autoencoders into a simulated supervised learning. By mixing both the reconstruction and prediction criteria, the training can be thought as a pseudo-semi-supervised learning. This makes the DAE loss a promising objective function. After regularization, the final training loss is:

$$L_{2,\alpha,\beta}(\mathbf{x},\tilde{\mathbf{x}}) = \alpha \left(\sum_{j \in \mathcal{C}(\tilde{\mathbf{x}}) \cap \mathcal{K}(\mathbf{x})} [nn(\tilde{\mathbf{x}})_j - x_j]^2 \right) + \beta \left(\sum_{j \notin \mathcal{C}(\tilde{\mathbf{x}}) \cap \mathcal{K}(\mathbf{x})} [nn(\tilde{\mathbf{x}})_j - x_j]^2 \right) + \lambda |\mathbf{W}|^2_{Fro},$$

where $\mathcal{K}(\mathbf{x})$ are the indices of known values of $\mathbf{x}$, $\mathbf{W}$ is the flatten vector of weights and λ is the regularization hyperparameter. The full forward/backward process is explained in Figure 1. Importantly, Autoencoders with sparse inputs differs from sparse-Autoencoders (Lee et al., 2006) or Dropout regularization (Srivastava et al., 2014) in the sense that Sparse Autoencoders and Droupout inhibit the hidden neurons while inputs/outputs are actually known.

3.2. Autoencoder and Low Rank Matrix Factorization

Autoencoders are actually strongly linked with Matrix Factorization. For an Autoencoder with only one hidden layer and no output transfer function, the response of the network can be written as

$$nn(\mathbf{x}) = \mathbf{W_2}\, \sigma(\mathbf{W_1 x} + \mathbf{b_1}) + \mathbf{b_2},$$

where $\mathbf{W_1}, \mathbf{W_2}$ are the weights matrices and $\mathbf{b_1}, \mathbf{b_2}$ the bias terms. Up to the bias term $\mathbf{b_2}$, if we replace $\mathbf{x}$ by the representation $\mathbf{u}_i$ of the user i, we recover a predicted vector $\hat{\mathbf{u}}_i$ of the form $\mathbf{V}\bar{\mathbf{u}}_\mathbf{i}$:

$$\hat{\mathbf{u}}_i = nn(\mathbf{u_i}) = \underbrace{\mathbf{W_2}}_{\mathbf{V}}\ \underbrace{\sigma(\mathbf{W_1 u_i} + \mathbf{b_1})}_{\bar{\mathbf{u}}_\mathbf{i}} + \mathbf{b_2}.$$

Symmetrically, $\hat{\mathbf{v}}_j$ has the form $\mathbf{U}\bar{\mathbf{v}}_\mathbf{j}$:

$$\hat{\mathbf{v}}_j = nn(\mathbf{v_j}) = \underbrace{\mathbf{W'_2}}_{\mathbf{U}}\ \underbrace{\sigma(\mathbf{W'_1 v_j} + \mathbf{b'_1})}_{\bar{\mathbf{v}}_\mathbf{j}} + \mathbf{b'_2}.$$

For the Matrix Factorization by ALS ; $\hat{\mathbf{R}}$ is iteratively built by solving for each row i of $\mathbf{U}$ (resp. column j of $\mathbf{V^T}$) a penalized least square regression using the known values of the i^{th} row of $\mathbf{R}$ (resp. j^{th} column of $\mathbf{R}$) as observations of a scalar product in dimension k of $\bar{\mathbf{u}}_\mathbf{i}$ and a column of $\mathbf{V^T}$ (resp $\bar{\mathbf{v}}_\mathbf{i}$ and a row of $\mathbf{U}$). An Autoencoder replaces the regularization introduced by the alternate scheme by a projection in dimension k composed with the non linearity σ. This process corresponds to a non linear matrix factorization.

Note that CFN breaks the symmetry between $\mathbf{U}$ and $\mathbf{V}$. For example, while Matrix Factorization approaches learn both $\mathbf{U}$ and $\mathbf{V}$, U-CFN learns $\mathbf{V}$ and only indirectly learns $\mathbf{U}$: U-CFN targets the function to build $\bar{\mathbf{u}}_\mathbf{i}$ whatever the row $\mathbf{u}_\mathbf{i}$. A direct benefit is that the learned Autoencoder is able to fill in every vector $\mathbf{u}_\mathbf{i}$, even if that vector was not in the training data.

Finally, both non-linear decompositions on rows and columns are done independently, which means that the matrix $\mathbf{V}$ learned by U-CFN from rows can differ from the concatenation of vectors $\bar{\mathbf{v}}_\mathbf{j}$ predicted by V-CFN from columns.

4. Integrating side information

Collaborative Filtering only relies on the feedback of users regarding a set of items. When additional information is available for the users and the items, this can sound restrictive. One would think that adding more information can help in several ways: increasing the prediction accuracy, speeding up the training, increasing the robustness of the model, etc. Furthermore, pure Collaborative Filtering suffers from the cold start problem: when very little information is available on an item, Collaborative Filtering will have difficulties recommending it. When bad recommendation are provided, the probability to receive valuable feedback is lowered leading to a vicious circle for new items.

A common way to tackle this problem is to add some side information to ensure a better initialization of the system. This is known in the recommendation community as hybridization.

The simplest approach to integrate side information is to append additional user/item bias to the rating prediction (Koren et al., 2009):

$$\hat{r}_{ij} = \bar{\mathbf{u}}_i^T \bar{\mathbf{v}}_j + b_{u,i} + b_{u,j} + b',$$

where $b_{u,i}$, $b_{u,j}$, b' are respectively the user, item, and global bias of the Matrix Factorization. Computing these bias can be done through hand-crafted engineering or Collaborative Filtering technique. For instance, one method is to extend the dense feature vectors by directly appending side information on them (Porteous & A. U. Asuncion, 2010). Therefore, the estimated rating is computed by:

$$\begin{aligned}\hat{r}_{ij} &= \{\bar{\mathbf{u}}_i, \mathbf{x}_i\} \otimes \{\bar{\mathbf{v}}_j, \mathbf{y}_j\} \\ &\stackrel{\text{def}}{=} \bar{\mathbf{u}}_{[1:k],i}^T \bar{\mathbf{v}}_{[1:k],j} + \underbrace{\bar{\mathbf{u}}_{[k+1:k+p],i}^T \mathbf{y}_j}_{b_{v,j}} + \underbrace{\bar{\mathbf{v}}_{[k+1:k+p],j}^T \mathbf{x}_i}_{b_{u,i}},\end{aligned}$$

where $\mathbf{x}_i \in \mathbb{R}^P$ and $\mathbf{y}_j \in \mathbb{R}^Q$ are respectively a vector representation of side information for the user and the item. If we decompose the previous scalar product, $\bar{\mathbf{u}}_{[1:k],i}^T \bar{\mathbf{v}}_{[1:k],j}$ is the classic Matrix Factorization term, $\bar{\mathbf{u}}_{[k+1:k+p],i}^T \mathbf{y}_j$ is the linear regression of the dense user representation and the item side information, $\bar{\mathbf{v}}_{[k+1:k+p],j}^T \mathbf{x}_j$ is the linear regression of the dense item representation and the user side information. For users, they can encode the genre, the age, job, circle of friends and so on. For items, they can be tags, a bag of words etc.

Unfortunately, those methods cannot be directly applied to Neural Networks because Autoencoders optimize $\mathbf{U}$ and $\mathbf{V}$ independently. New strategies must be designed to incorporate side information. One notable example for bitext word alignment was recently made by (Ammar et al., 2014).

In our case, the first idea would be to append the side information to the sparse input vector. For simplicity purpose, the next equations will only focus on shallow U-Autoencoders with no output transfer functions. Yet, this can be extended to more complex Networks and V-Autoencoders. Therefore, we would get:

$$\begin{aligned}nn(\{\mathbf{u}_i, \mathbf{x}_i\}) &= \mathbf{V}\, \sigma(\mathbf{W}_1' \{\mathbf{u}_i, \mathbf{x}_i\} + \mathbf{b_1}) + \mathbf{b_2} \\ &= \hat{\mathbf{u}}_i,\end{aligned}$$

where $\mathbf{W}_1' \in \mathbb{R}^{k \times (n+p)}$ is a weight matrix . When no previous rating exist, it enables the Neural Networks to have at an input to predict new ratings. With this scenario, side information is assimilated to pseudo-ratings that will always exist for every items. However, when the dimension of the Neural Network input is far greater than the dimension of the side information, the Autoencoder may have difficulties to use it efficiently.

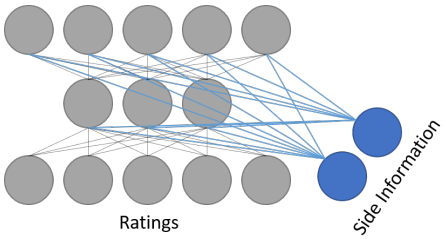

Figure 2. Integrating side information. The Network has two inputs: the classic Autoencoder rating input and a side information input. Side information is wired to every neurons in the Network.

Yet, common Matrix Factorization would append side information to dense feature representations $\{\bar{\mathbf{u}}_i, \mathbf{x}_i\}$ rather than sparse feature representation as we just proposed $\{\mathbf{u}_i, \mathbf{x}_i\}$. Thus, one way to reproduce this idea is to inject the side information to every layer inputs of the Network.

$$\begin{aligned}nn(\{\mathbf{u}_i, \mathbf{x}_i\}) &= \mathbf{V}' \{\overbrace{\sigma(\mathbf{W}_1' \{\mathbf{u}_i, \mathbf{x}_i\} + \mathbf{b_1})}^{\bar{u}'}, \mathbf{x}_i\} + \mathbf{b_2} \\ &= \mathbf{V}' \{\bar{\mathbf{u}}_i', \mathbf{x}_i\} + \mathbf{b_2} \\ &= \mathbf{V}'_{[1:k]} \bar{\mathbf{u}}_i' + \underbrace{\mathbf{V}'_{[k+1:k+p]} \mathbf{x}_i}_{\mathbf{b}_{u,i}} + \mathbf{b_2},\end{aligned}$$

where $\mathbf{V}' \in \mathbb{R}^{(n \times k+p)}$ is a weight matrix, $\mathbf{V}'_{[1:k]} \in \mathbb{R}^{n \times k}$, $\mathbf{V}'_{[k+1:k+p]} \in \mathbb{R}^{n \times p}$ are respectively the submatrices of $\mathbf{V}'$ that contain the columns from 1 to k and $k+1$ to $k+p$.

By injecting the side information in every layer, the dynamic Autoencoders representation is forced to integrate this new data. In addition, the output layer computes a linear regression of the side information. However, we do not want the side information to overstep the dense rating representation. Thus, we enforced the following constraint. The dimension of the sparse input must be greater than the dimension of the Autoencoder bottleneck which must be greater than the dimension of the side information [1]:

$$P << K_u << N \quad \text{and} \quad Q << K_v << M.$$

We finally obtain an Autoencoder which can incorporate side information and be trained through backpropagation.

[1] When side information is sparse, the dimension of the side information can be assimilated to the number of non-zero parameters

5. Experiments

5.1. Benchmark Models

We benchmark CFN with two SVD techniques that are broadly used in the industry.

Alternating Least Squares with Weighted-λ-Regularization (ALS-WR) (Zhou et al., 2008) solves the low-rank matrix factorization problem by alternatively fixing $\mathbf{U}$ and $\mathbf{V}$ and solving the resulting linear problem. Tikhonov regularization is used as it empirically provides excellent results:

$$L_2 = \sum_{(i,j)\in\mathcal{K}(\mathbf{R})} (r_{ij} - \mathbf{u}_i^T \mathbf{v}_j)^2 + \lambda(n_i \|\mathbf{u}_i\|_{Fro}^2 + n_j \|\mathbf{v}_j\|_{Fro}^2),$$

where n_i is the number of rating for the i^{th} user and n_j is the number of rating for the j^{th} item. Experiments are run with the Apache Mahout Software. [2]

SVDFeature (Chen et al., 2012) is a Machine Learning Toolkit for feature-based Collaborative Filtering. He won the KDD Cup for two consecutive years. Ratings are given by the following equation:

$$\hat{r} = \left(\sum_p^{N+P} x_p b_p^{(u)} + \sum_q^{M+Q} y_q b_q^{(v)} + \sum_r^{\|R\|} z_r b_r^{(g)} \right) + \left(\sum_p^{N+P} x_p \mathbf{u}_p \right)^T \left(\sum_q^{M+Q} y_q \mathbf{v}_q \right)$$

where $\mathbf{b}^{(u)} \in \mathbb{R}^{N+P}$, $\mathbf{b}^{(i)} \in \mathbb{R}^{M+Q}$, $\mathbf{b}^{(g)} \in \mathbb{R}^{\|R\|}$ are the the side information bias, and $\mathbf{U} \in \mathbb{R}^{N+P\times K}$, $\mathbf{V} \in \mathbb{R}^{M+Q\times K}$ encode the latent factors. The model parameters are computed by gradient descent.

5.2. Data

Experiments are conducted on MovieLens and Douban datasets.

The MovieLens-1M, MovieLens-10M and MovieLens-20M datasets respectively provide 1/10/20 millions discrete ratings from 6/72/138 thousands users on 4/10/27 thousands movies. Side information for MovieLens-1M is the age, sex and gender of the user and the movie category (action, thriller etc.). Side information for MovieLens-10/20M is a matrix of tags $\mathbf{T}$ where T_{ij} is the occurrence of the j^{th} tag for the i^{th} movie and the movie category. No side information is provided for users.

The Douban dataset (Ma et al., 2011) provides 17 million discrete ratings from 129 thousands users on 58 thousands movies. Side information is the bi-directional user/friend relations for the user. The user/friend relation are treated like the matrix of tags from MovieLens. No side information is provided for items.

[2] http://mahout.apache.org/

Preprocessing For each of these datasets, the full dataset is considered and the ratings are normalized from -1 to 1. We split them into random 80%-20% train-test datasets and inputs are unbiased before the training process: denoting μ the mean over the training set, b_{u_i} the mean of the i^{th} user and b_{v_i} the mean of the v^{th} item, SVD algorithms, U-CFN and V-CFN respectively learn from $r_{ij}^{unbiased} = r_{ij} + \mu - b_{u_i} - b_{v_j}$, $r_{ij}^{unbiased} = r_{ij} - b_{u_i}$ and $r_{ij}^{unbiased} = r_{ij} - b_{v_i}$.

Postprocessing The bias computed on the training set is added back while evaluating the final RMSE.

Side Information In order to enforce the side information constraint, $Q \ll K_v \ll M$, Principal Component Analysis is performed on the matrix of tags. We keep the 50 greatest eigenvectors[3] and normalize them by the square root of their respective eigenvalue: given $\mathbf{T} = \mathbf{PDQ}^T$ with $\mathbf{D}$ the diagonal matrix of eigenvalues sorted in descending order, the movie tags are represented by $\mathbf{Y} = \mathbf{P}_{J\times K'} \mathbf{D}_{K'\times K'}^{0.5}$ with K' the number of kept eigenvectors. Binary representation such as the movie category is then concatenated to $\mathbf{Y}$.

5.3. Error Function

Two Mean Square Errors (MSE) co-exist for Autoencoders and one must be careful to use the right estimator for benchmarking. The classic Autoencoder loss estimates the quality of the input reconstruction:

$$MSE_{rec}(\mathbf{X_{tr}}) = \frac{1}{|R_{tr}|} \sum_{\mathbf{x}\in\mathbf{X_{tr}}} \sum_{k\in\mathcal{K}(\mathbf{x})} [nn(\mathbf{x})_k - x_k]^2,$$

where $|R_{tr}|$ is the number of ratings in the training dataset.

In Collaborative Filtering context, while this loss provides useful information during the training phase, the quality of the learned Autoencoder is given by its prediction accuracy for unknown inputs. This prediction accuracy is summed up couples $(\mathbf{x_{te}}, \mathbf{x_{tr}})$ of training-testing examples:

$$MSE_{pred}(\mathbf{X_{te}}, \mathbf{X_{tr}}) = \frac{1}{|R_{te}|} \sum_{(\mathbf{x_{te}},\mathbf{x_{tr}})\in(\mathbf{X_{te}},\mathbf{X_{tr}})} \sum_{k\in\mathcal{K}(\mathbf{x_{te}})} [nn(\mathbf{x_{tr}})_k - x_{te,k}]^2.$$

We use MSE_{pred} to evaluate both the baselines and CFN.

[3] The number of eigenvalues is arbitrary selected. We do not focus on optimizing the quality of this representation.

Table 1. RMSE with a training ratio of 90%/10%. The ++ acronym is appended to algorithms with side information. When no side information is available, the N/A acronym is used. When no good results were found after two days of computation, the * character is used.

Algorithms	MovieLens-1M	MovieLens-10M	MovieLens-20M	Douban
ALS-WR	0.8526 ± 2.4e-3	0.7949 ± 1.8e-3	0.7864 ± 3.2e-3	0.7117 ± 3.3e-3
SVDFeature	0.8631 ± 2.5e-3	0.7907 ± 8.4e-4	*	*
U-CFN	0.8574 ± 2.4e-3	0.7954 ± 7.4e-4	0.7896 ± 1.4e-4	0.7049 ± 2.2e-4
U-CFN++	0.8572 ± 1.6e-3	N/A	N/A	0.7050 ± 1.2e-4
V-CFN	0.8388 ± 2.5e-3	0.7780 ± 5.4e-4	0.7669 ± 2.6e-4	**0.6911 ± 3.2e-4**
V-CFN++	**0.8377 ± 1.8e-3**	**0.7764 ± 6.3e-4**	**0.7762 ± 4.6e-4**	N/A

5.4. Training Settings

We train 4-layers Autoencoders for MovieLens-1M and 2-layers Autoencoders for MovieLens-10/20M and the Douban datasets. The first and second layers have from 500 to 700 hidden neurons. The decoding layers have the same dimension in the reverse order. Weights are initialized using the fan-in rule $\mathbf{W_{ij}} \sim U\left[-\frac{1}{\sqrt{n}}, -\frac{1}{\sqrt{n}}\right]$ (LeCun et al., 1998). Transfer functions are hyperbolic tangents. The Neural Network is optimized with stochastic backpropagation with minibatch of size 30 and a weight decay is added for regularization. Hyperparameters [4] are tuned by a simple genetic algorithm already used by (Teytaud et al., 2007) in a different context.

5.5. Results

Table 1 summarizes the RMSE on MovieLens and Douban datasets. Reported results are computed through k-fold cross-validation and confidence intervals correspond to a 95% range. To the best of our knowledge, the best results published regarding MovieLens-1M and MovieLens-10M are reported by both (Lee et al., 2013; Sedhain et al., 2015) with a final RMSE of 0.831 ± 0.003 and 0.782 ± 0.003. These scores are obtained with a training ratio of 90%/10% and without side information. (Kim & Choi, 2014) use a scalable version of BPMF with side information and they report respectively 0.844 and 0.6856 RMSE score on MovieLens-10M and Douban with a training ratio of 80%/20% while V-CFN++ returns 0.782 and 0.694 with the same experimental conditions.

5.6. Discussion

V-CFNs have excellent performance in our experiments for every dataset we run. It is competitive compared to the state-of-the-art Collaborative Filtering algorithms and clearly outperforms them for MovieLens-10M. For instance, recent works still reports a global RMSE above 0.8 even using some side information with MovieLens-10M (Kim & Choi, 2014; Kumar et al., 2014). The confidence interval of CFN is also low on large datasets which make him a robust algorithm for Collaborative Filtering.

In the experiments, V-CFN outperforms U-CFN. It suggests that the structure on the items is stronger than the one on users *i.e.* it is easier to guess tastes based on movies you liked than to find some users similar to you. Of course, the behavior could be different on some other data .

At first sight, the use of side information has a limited impact on the RMSE. This statement has to be mitigated: as the repartition of known entries in the dataset is not uniform, the estimates are biased towards users and items with a lot of ratings. For theses users and movies, the dataset already contains a lot of information, thus having some extra information will have a marginal effect. Users and items with few ratings should benefit more from some side information but the estimation biais hides them.

Table 3. RMSE computed by cluster of items sorted by their respective number of ratings on MovieLens-10M with a training ratio of 90%/10%. For instance, the first cluster contains the 20% of items with the lowest number of ratings. The provided figures are the mean reported through k-cross validation. The last cluster far outweigh other clusters and hide more subtle results.

Interval	V-CFN	V-CFN++	Improvement %
0.0-0.2	0.9568	0.9373	1.811
0.2-0.4	0.8746	0.8644	1.215
0.4-0.6	0.8501	0.8446	0.760
0.6-0.8	0.8130	0.8097	0.398
0.8-1.0	0.7675	0.7671	0.052
Full	0.7780	0.7764	0.206

In order to exhibit the utility of side information, we report in Table 3 the RMSE conditionally to the number of missing values for items. As expected, the fewer number of ratings for an item, the more important the side information. A more careful analysis of the RMSE improvement in this setting shows that the improvement is uniformly distributed over the users whatever their number of ratings. This corresponds to the fact that the available side information is only about items. This is very desirable for a real system: the ef-

[4] Hyperparameters used for the experiments are provided with the source code.

Table 2. MAE with a training ratio of 90%/10%.

Algorithms	MovieLens-1M	MovieLens-10M	MovieLens-20M	Douban
ALS-WR	0.6773 ± 2.2e-3	0.6199 ± 1.8e-4	0.6104 ± 5.3e-4	0.5669 ± 3.4e-4
U-CFN	0.6727 ± 2.3e-4	0.6089 ± 2.4e-4	0.6004 ± 1.2e-4	0.5561 ± 2.4e-4
U-CFN++	0.6725 ± 9.7e-4	N/A	N/A	0.5563 ± 1.8e-4
V-CFN	0.6564 ± 1.9e-3	0.5951 ± 1.3e-4	0.5824 ± 2.4e-4	**0.5442 ± 3.1e-4**
V-CFN++	**0.6548 ± 1.3e-3**	**0.5936 ± 3.6e-4**	**0.5817 ± 1.9e-4**	N/A

fective use of side information to the new items is crucial to deal with the flow of new products.

In the end, we trained V-CFN on MovieLens-10M with either the movie genre or the matrix of tags. Both side information increase the global RMSE by 0.1% and concatenating them increase the final score by a small margin of 0.15%. Therefore, V-CFN could handle the heterogeneity of side information. However, the U-CFN failed to use the friendship relationship to increase the RMSE.

6. Remarks

6.1. Source code

Torch is a powerful framework written in Lua to quickly prototype Neural Networks. It is a widely used (Facebook, Deep Mind) industry standard. However, Torch lacks some important basic tools to deal with sparse inputs. Thus, we develop several new modules to deal with DAE loss, sparse DAE loss and sparse inputs on both CPU and GPU. They can easily be plugged into existing code. An out-of-the-box tutorial is available to directly run the experiments. The code is freely available on Github and Luarocks. [5]

6.2. Scalability

One major problem that most Collaborative Filtering have to resolve is scalability since dataset often have hundred of thousands users and items. An efficient algorithm must be trained in a reasonable amount of time and provide quick feedback during evaluation time.

Recent advances in GPU computation managed to reduce the training time of Neural Networks by several orders of magnitude. However, Collaborative Filtering deals with sparse data and GPUs are designed to perform well on dense data. (Salakhutdinov et al., 2007; Sedhain et al., 2015) face this sparsity constraint by building small dense Networks with shared weights. Yet, this approach may lead to important synchronisation latencies. In our case, we tackle the issue by selectively densifying the inputs just before sending them to the GPUs cores without modification of the result of the computation. It introduces an overhead on the computational complexity but this implementation allows the GPUs to work at their full strength. In practice, vectorial operations overtake the extra cost. Such approach is an efficient strategy to handle sparse data which achieves a balance between memory footprint and computational time. We are able to train Large Neural Networks within a few minutes as shown in Table 4. At the time of writing, alternative strategies to train networks with sparse inputs on GPUs are under development.

[5] https://github.com/*******

6.3. Future Works

Implicit feedback may greatly enhance the quality of Collaborative Filtering algorithms (Koren et al., 2009; Rendle, 2010). For instance, Implicit feedback would be incorporated to CFN by feeding the Network with an additional binary input. By doing so, (Salakhutdinov et al., 2007) enhance the quality of prediction for Restricted Boltzmann Machine on the Netflix Dataset. Additionally, Content Based Technique with Deep learning such as (den Oord et al., 2013; Wang et al., 2014b) would be plugged to CFN. The idea would be to train a joint Network that would directly link the raw item features to the ratings such as music, pictures or word representations. As a different topic, V-CFN and U-CFN does not always report the same type of errors. This is more likely to happen when they are fed with side information. One interesting work would be to combine a suitable Network that mix them both. Finally, other metrics exist to estimate the quality of Collaborative Filtering to fit other real-world constraints. Normalized Discounted Cumulative Gain (Järvelin & Kekäläinen, 2002) or F-score are often sometimes preferred to RMSE/MAE and should be benchmarked.

7. Conclusion

In this paper, we have introduced a Neural Network architecture, aka CFN, to perform Collaborative Filtering with side information. Contrary to other attempts with Neural Networks, this joint Network integrate side information and learn a non-linear representation of users or items into a unique Neural Network. This approach manages to both beats state of the art results in CF and ease the cold start problem on the MovieLens and Douban datasets. CFN is also scalable and robust to deal with large size dataset. We made several claims that Autoencoders are closely linked

Table 4. Training time and memory footprint for a 2-layers CFN without side information. The GPU is a standard GTX 980. $Time$ is the average training duration (around 20 epochs/networks). Parameters are the weight and bias matrices. Memory is retrieved by the GPU driver during the training. It includes the dataset, the model parameters and the training buffer. Although the memory footprint highly depends on the implementation, it provides a good order of magnitude. Adding side information would increase by fewer than 5% the final time and memory footprint.

Dataset	Type	# Param	Time	Memory
MLens-1M	V	8M	2m03s	250MiB
MLens-10M	V	100M	18m34s	1532MiB
MLens-20M	V	194M	34m45s	2905MiB
MLens-1M	U	5M	7m17s	262MiB
MLens-10M	U	15M	34m51s	543MiB
MLens-20M	U	38M	59m35s	1044Mib

to low-rank Matrix Factorization in Collaborative Filtering. Finally, a reusable source code is provided in Torch and hyperparameters are provided to reproduce the results.

References

Adams, R. P. and Murray, G. E. Dahland I. Incorporating side information in probabilistic matrix factorization with gaussian processes. *arXiv preprint arXiv:1003.4944*, 2010.

Ammar, W., Dyer, C., and Smith, N. A. Conditional random field autoencoders for unsupervised structured prediction. In *Advances in Neural Information Processing Systems*, pp. 3311–3319, 2014.

Bishop, C. M. *Neural networks for pattern recognition*. Oxford university press, 1995.

Chen, T., Zhang, W., Lu, Q., Chen, K., Zheng, Z., and Yong, Y. Svdfeature: a toolkit for feature-based collaborative filtering. *The Journal of Machine Learning Research*, 13(1):3619–3622, 2012.

den Oord, A. Van, Dieleman, S., and Schrauwen, B. Deep content-based music recommendation. In *Advances in Neural Information Processing Systems*, pp. 2643–2651, 2013.

Dziugaite, G.K. and Roy, D.M. Neural network matrix factorization. *arXiv preprint arXiv:1511.06443*, 2015.

Glorot, X. and Bengio, Y. Understanding the difficulty of training deep feedforward neural networks. In *International conference on artificial intelligence and statistics*, pp. 249–256, 2010.

Glorot, X., Bordes, A., and Bengio, Y. Deep sparse rectifier neural networks. In *International Conference on Artificial Intelligence and Statistics*, pp. 315–323, 2011.

Gomez-Uribe, C. and Hunt, N. The netflix recommender system: Algorithms, business value, and innovation. *ACM Trans. Manage. Inf. Syst.*, 6(4):13:1–13:19, 2015.

Järvelin, K. and Kekäläinen, K. Cumulated gain-based evaluation of ir techniques. *ACM Transactions on Information Systems (TOIS)*, 20(4):422–446, 2002.

Kim, Yong-Deok and Choi, Seungjin. Scalable variational bayesian matrix factorization with side information. In *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS), Reykjavik, Iceland*, 2014.

Koren, Y., Bell, R., and Volinsky, C. Matrix factorization techniques for recommender systems. *Computer*, (8): 30–37, 2009.

Kramer, M. A. Nonlinear principal component analysis using autoassociative neural networks. *AIChE journal*, 37 (2):233–243, 1991.

Kumar, R., Verma, B. K., and Rastogi, S. S. Social popularity based svd++ recommender system. *International Journal of Computer Applications*, 87(14), 2014.

Lawrence, N.D. and Urtasun, R. Non-linear matrix factorization with gaussian processes. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pp. 601–608. ACM, 2009.

LeCun, Y., Bengio, Y., and Hinton, G. Deep learning. *Nature*, 521(7553):436–444, 2015.

LeCun, Y.A., Bottou, L., Orr, G.B., and Muller, K.R. Efficient backprop. In *Neural networks: Tricks of the trade*, pp. 9–48. Springer, 1998.

Lee, H., Battle, A., Raina, R., and Ng, A.Y. Efficient sparse coding algorithms. In *Advances in neural information processing systems*, pp. 801–808, 2006.

Lee, J., Sun, M., and Lebanon, G. A comparative study of collaborative filtering algorithms. *arXiv preprint arXiv:1205.3193*, 2012.

Lee, J., Kim, S., Lebanon, G., and Singerm, Y. Local low-rank matrix approximation. In *Proceedings of The 30th International Conference on Machine Learning*, pp. 82–90, 2013.

Li, S., Kawale, J., and Fu, Y. Deep collaborative filtering via marginalized denoising auto-encoder. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, pp. 811–820. ACM, 2015.

Lops, P., Gemmis, M. De, and Semeraro, G. Content-based recommender systems: State of the art and trends. In *Recommender systems handbook*, pp. 73–105. Springer, 2011.

Ma, H., Zhou, D., Liu, C., Lyu, M. R., and King, I. Recommender systems with social regularization. In *Proceedings of the fourth ACM international conference on Web search and data mining*, WSDM '11, pp. 287–296, Hong Kong, China, 2011.

Miranda, V., Krstulovic, J., Keko, H., Moreira, C., and Pereira, J. Reconstructing Missing Data in State Estimation With Autoencoders. *IEEE Transactions on Power Systems*, 27(2):604–611, 2012. ISSN 0885-8950.

Mnih, A. and Salakhutdinov, R. Probabilistic matrix factorization. In *Advances in neural information processing systems*, pp. 1257–1264, 2007.

Porteous, I. and A. U. Asuncion, M. Welling. Bayesian matrix factorization with side information and dirichlet process mixtures. In *AAAI*, 2010.

Rendle, S. Factorization machines. In *Data Mining (ICDM), 2010 IEEE 10th International Conference on*, pp. 995–1000. IEEE, 2010.

Salakhutdinov, R. and Mnih, A. Bayesian probabilistic matrix factorization using markov chain monte carlo. In *Proceedings of the 25th international conference on Machine learning*, pp. 880–887. ACM, 2008.

Salakhutdinov, R., Mnih, A., and Hinton, G. Restricted boltzmann machines for collaborative filtering. In *Proceedings of the 24th international conference on Machine learning*, pp. 791–798. ACM, 2007.

Sedhain, S., Menon, A. K., Sanner, S., and Xie, L. Autorec: Autoencoders meet collaborative filtering. In *Proceedings of the 24th International Conference on World Wide Web Companion*, pp. 111–112. International World Wide Web Conferences Steering Committee, 2015.

Srivastava, N, Hinton, G., Krizhevsk, A., Sutskever, I., and Salakhutdinov, R. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

Strub, F. and Mary, J. Collaborative Filtering with Stacked Denoising AutoEncoders and Sparse Inputs. In *NIPS Workshop on Machine Learning for eCommerce*, Montreal, Canada, 2015.

Teytaud, O., S., and Mary, J. Active learning in regression, with application to stochastic dynamic programming. In International Conference On Informatics in Control, Automation and Robotics (eds.), *ICINCO and CAP*, pp. 373–386, 2007.

Tresp, V., Ahmad, S., and Neuneier, R. Training Neural Networks with Deficient Data. *Advances in Neural Information Processing Systems 6*, pp. 128–135, 1994. doi: 10.1.1.23.6971.

Truyen, T. T., Phung, D.Q., and Venkatesh, S. Ordinal boltzmann machines for collaborative filtering. In *Proceedings of the Twenty-fifth Conference on Uncertainty in Artificial Intelligence*, pp. 548–556. AUAI Press, 2009.

Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., and Manzagol, P. Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion. *Journal of Machine Learning Research*, 11(3):3371–3408, 2010. ISSN 15324435. doi: 10.1111/1467-8535.00290.

Wang, H., Wang, N., and Yeung, D. Y. Collaborative deep learning for recommender systems. *arXiv preprint arXiv:1409.2944*, 2014a.

Wang, H., Wang, N., and Yeung, D. Y. Improving content-based and hybrid music recommendation using deep learning. In *Proceedings of the ACM International Conference on Multimedia*, pp. 627–636. ACM, 2014b.

Zhou, Y., Wilkinson, D., Schreiber, R., and Pan, R. Large-scale parallel collaborative filtering for the netflix prize. In *Algorithmic Aspects in Information and Management*, pp. 337–348. Springer, 2008.

ACKNOWLEDGEMENTS

The authors would like to acknowledge the stimulating environment provided by SequeL research group, Inria and CRIStAL. This work was supported by French Ministry of Higher Education and Research, by CPER Nord-Pas de Calais/FEDER DATA Advanced data science and technologies 2015-2020, the Projet CHIST-ERA IGLU and by FUI Hermès. Experiments were carried out using Grid'5000 tested, supported by Inria, CNRS, RENATER and several universities as well as other organizations.