Hybrid Recommender System based on Autoencoders

Florian Strub
Univ. Lille, CNRS,
Centrale Lille, Inria
UMR 9189 - CRIStAL
F-59000 Lille, France
florian.strub@inria.fr

Romaric Gaudel
Univ. Lille, CNRS,
Centrale Lille, Inria
UMR 9189 - CRIStAL
F-59000 Lille, France
romaric.gaudel@inria.fr

Jérémie Mary
Univ. Lille, CNRS,
Centrale Lille, Inria
UMR 9189 - CRIStAL
F-59000 Lille, France
jeremie.mary@inria.fr

ABSTRACT

A standard model for Recommender Systems is the Matrix Completion setting: given partially known matrix of ratings given by users (rows) to items (columns), infer the unknown ratings. In the last decades, few attempts where done to handle that objective with Neural Networks, but recently an architecture based on Autoencoders proved to be a promising approach. In current paper, we enhanced that architecture (i) by using a loss function adapted to input data with missing values, and (ii) by incorporating side information. The experiments demonstrate that while side information only slightly improve the test error averaged on all users/items, it has more impact on cold users/items.

1. INTRODUCTION

Recommendation systems (RS) advise users on which items (movies, musics, books, etc.) they are more likely to be interested in. A good RS may dramatically increase the amount of sales of a firm or retain customers. For instance, 80% of movies watched on Netflix come from the RS of the company [8]. 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* (CBF) and *Collaborative Filtering* (CF).

CBF 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. On the other side, 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 CBF because it wins the agnostic vs. studied contest: CF only relies on the ratings of the users while CBF requires advanced engineering on items to well perform [20].

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 [10] or Regularized Alternating Least Square algorithm [39]. However, these methods are linear and cannot catch subtle factors. Newer algorithms were explored to face those constraints such as Factorization Machines [25]. More recent works combine several low-rank matrices such as Local Low Rank Matrix Approximation [16] or WEMAREC [3] to enhance the recommendation.

Another limitation of CF is known as the *cold start* problem: how to recommend an item to a user when no rating exists for either the user or the item? To overcome this issue, one idea is to build a hybrid model mixing CF and CBF where side information is integrated into the training process. The goal is to supplant the lack of ratings through side information. A successful approach [1, 24] extends the Bayesian Probabilistic Matrix Factorization Framework [26] to integrate side information. However, recent algorithms outperform them in the general case [17].

In this paper we introduce a CF approach based on Stacked Denoising Autoencoders [35, 40] 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 [27, 28, 30, 5, 38], our framework integrates the 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 fixes the setting and gives state of the art on related approaches. Then, our model is described in Sec. 3. Finally, experimental results are given and discussed in Sec. 4 and Sec. 5 discusses algorithmic aspects.

2. PRELIMINARIES

2.1 Matrix Completion

A standard setting for CF is Matrix Completion [10]. Given N users and M items, the rating r_{ij} is the rating given by the ith user for the jth item. It entails a matrix of ratings $\mathbf{R} \in \mathbb{R}^{N \times M}$, for which only a few entries are known. The

goal of Matrix Completion is to infer the unknown value. Namely, the algorithm returns a matrix $\widehat{\mathbf{R}} \in \mathbb{R}^{N \times M}$ which hopefully minimizes the reconstruction error

$$\mathcal{L}(\mathbf{R}, \widehat{\mathbf{R}}) = \sum_{(i,j) \notin \mathcal{K}(\mathbf{R})} (r_{ij} - \hat{r}_{ij})^2,$$

where $\mathcal{K}(\mathbf{R})$ is the set of indices of known ratings of $\mathbf{R}$.

2.2 Denoising Autoencoders

The proposed approach builds upon Autoencoders which are feed-forward Neural Networks popularized by Kramer [11]. They are unsupervised Networks where the output of the Network aims at reconstructing the initial input. The Network is trained by back-propagating the squared error loss on the reconstruction. When the network limits itself to one hidden layer, its output is given by

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

with $\mathbf{x} \in \mathbb{R}^N$ the input, $\mathbf{W_1} \in \mathbb{R}^{k \times N}$ and $\mathbf{W_2} \in \mathbb{R}^{N \times k}$ 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. The size $k \ll N$ of the hidden layer is also known as the *Autoencoder's bottleneck.*

Recent work in Deep Learning advocates to stack pre-trained encoders to initialize Deep Neural Networks [6]. 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 may degenerate into identity Networks and they fail to learn the latent relationship between data. [35] 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 of the input. Another method is to mask a random fraction 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):

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

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

where $\tilde{\mathbf{x}} \in \mathbb{R}^N$ is a corrupted version of the input $\mathbf{x}$, $\mathcal{C}$ is the set of corrupted elements in $\tilde{\mathbf{x}}$, and $nn(\mathbf{x})_j$ is the jth output of the Network while fed with $\mathbf{x}$.

2.3 Related Work

Neural Networks have attracted little attention in the CF community. In a preliminary work, [27] tackled the Netflix challenge using Restricted Boltzmann Machines but little published work had follow [33]. While Deep Learning has tremendous success in image and speech recognition [13], 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 [13] which makes them a promising tool for CF. [28, 30, 5] 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 [28] 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 CBF: Neural Networks are first trained to learn a feature representation from the item which is then processed by a CF approach such as Probabilistic Matrix Factorization [23] to provide the final rating. For instance, [7, 36] respectively auto-encode bag-of-words from restaurant reviews and movie plots, [19] auto-encode heterogeneous side information from users and items. Finally, [34, 37] 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 use the following notations:

- $\mathbf{u_i}$, $\mathbf{v_j}$ are the partially known rows/columns of $\mathbf{R}$;

- $\tilde{\mathbf{u}}_i$, $\tilde{\mathbf{v}}_j$ are corrupted versions of $\mathbf{u_i}$, $\mathbf{v_j}$;

- $\hat{\mathbf{u}}_i$, $\hat{\mathbf{v}}_j$ are rows/columns of $\widehat{\mathbf{R}}$, which is an estimate any entry of $\mathbf{R}$.

3. AUTOENCODERS AND COLLABORATIVE FILTERING

We propose to turn the sparse vectors $\mathbf{u}_i/\mathbf{v}_j$, into dense vectors $\hat{\mathbf{u}}_i/\hat{\mathbf{v}}_j$ with Autoencoders. To do so, we need to define two types of Autoencoders:

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

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

Note that CF is part of the few applications requiring to infer missing values, and not only to compress the available information.

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 [32, 2]. In our case, we want the Autoencoder to handle this prediction issue by itself. Such problems have already been studied in industry [22] 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, while the error is back-propagated for actual zero 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 [27, 28]. 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:

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

$$\beta \left(\sum_{j \in \mathcal{K}(\mathbf{x}) \setminus \mathcal{C}(\tilde{\mathbf{x}})} [nn(\tilde{\mathbf{x}})_j - x_j]^2 \right) + \lambda \|\mathbf{W}\|_F^2,$$

where $\mathcal{K}(\mathbf{x})$ are the indices of known values of $\mathbf{x}$, $\mathbf{W}$ is the flatten vector of weights of the Network and λ is the regularization hyperparameter. This loss is optimized thanks to standard forward/backward process on mini-batches. Importantly, Autoencoders with sparse inputs differs from sparse-Autoencoders [15] or Dropout regularization [29] in the sense that Sparse Autoencoders and Droupout inhibit the hidden neurons for regularization purpose. Every inputs/outputs are also known.

3.2 Integrating Side Information

From the time being U/V-CFN only relies on the feedback of users regarding a set of items. Let's now incorporate additional information about the users and the items. We will show that these information help in several ways: increase the prediction accuracy, speed up the training, increase the robustness of the model, etc. Last but not least, incorporating side information is a well-known approach to tackle the cold start problem: when very little information is available on a user/item, Collaborative Filtering will have difficulties to infer its ratings.

Instead of only adding the side information to the first layer of the Autoencoder, we propose to inject that information to every layer inputs of the Network. As an example, the model of U-CFN becomes

$$nn(\{\mathbf{u}_i, \mathbf{z}_i\}) = \sigma(\mathbf{W}_2'\{\sigma(\mathbf{W}_1'\{\mathbf{u}_i, \mathbf{z}_i\} + \mathbf{b_1}), \mathbf{z}_i\} + \mathbf{b_2}),$$

where $\mathbf{z}_i \in \mathbb{R}^P$ is the vector of side informations, $\mathbf{W}_1' \in \mathbb{R}^{k \times (N+P)}$ and $\mathbf{W}_2' \in \mathbb{R}^{N \times (k+P)}$ are the weight matrices, $\{\mathbf{x}, \mathbf{z}_i\}$ represents the concatenation of both vectors $\mathbf{x}$ and $\mathbf{z}_i$, and biases b_1 and b_2 respectively belong to $\mathbb{R}^{N+P}$ and $\mathbb{R}^{k+P}$.

By injecting the side information in every layer, the dynamic Autoencoders representation is forced to integrate this new data. However, to avoid side information to overstep the dense rating representation, we enforce the following constraints. 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]. Therefore, we get $P \ll k \ll N$ and $Q \ll k \ll M$.

4. EXPERIMENTS

4.1 Benchmark Models

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

Alternating Least Squares with Weighted-λ-Regularization (ALS-WR) [39] solves a low-rank matrix factorization problem by alternatively fixing $\mathbf{U}$ and $\mathbf{V}$ and solving the resulting linear problem. Experiments are run with the Apache Mahout Software[2].

SVDFeature [4] 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.

4.2 Data

Experiments are conducted on MovieLens and Douban datasets. The MovieLens-1M, MovieLens-10M and Movie-Lens-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 jth tag for the ith movie and the movie category. No side information is provided for users. The Douban dataset [21] 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.

Preprocessing.

For each dataset, the full dataset is considered and the ratings are normalized from -1 to 1. We split the dataset into random 90%-10% train-test datasets and inputs are unbiased before the training process: denoting μ the mean over

---

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

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

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 results were too low 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

the training set, b_i the mean of the ith user and b_j the mean of the jth item, U-CFN and V-CFN respectively learn from $r_{ij}^{unbiased} = r_{ij} - b_i$ and $r_{ij}^{unbiased} = r_{ij} - b_j$. The bias computed on the training set is added back while evaluating the learned matrix.

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 $\mathbf{T} = \mathbf{P}\mathbf{D}\mathbf{Q}^T$ 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 $\mathbf{Y}$.

4.3 Error Function

The algorithms are compared based on their respective *Root Mean Square Error* (RMSE) on test data. Denoting $\mathbf{R_{test}}$ the matrix of test ratings and $\widehat{\mathbf{R}}$ the full matrix returned by the learning algorithm, the RMSE is:

$$RMSE(\widehat{\mathbf{R}}, \mathbf{R_{test}}) =$$
$$\sqrt{\frac{1}{|\mathcal{K}(R_{test})|} \sum_{(i,j) \in \mathcal{K}(R_{test})} (r_{test,ij} - \hat{r}_{ij})^2},$$

where $|\mathcal{K}(R_{test})|$ is the number of ratings in the testing dataset. Note that for the sake of fair comparison, in the case of Autoencoders $\widehat{\mathbf{R}}$ is computed by feeding the network with **training** data. As such, $\hat{r}_{ij}$ stands for $nn(\mathbf{u}_{train,i})_j$ for U-CFN, and $nn(\mathbf{v}_{train,j})_i$ for V-CFN.

4.4 Training Settings

We train 2-layers Autoencoders for MovieLens-1/10/20M and the Douban datasets. The layers have from 500 to 700 hidden neurons. Weights are initialized using the fan-in rule [14]: $\mathbf{W_{ij}} \sim \mathcal{U}\left[-\frac{1}{\sqrt{n}}, \frac{1}{\sqrt{n}}\right]$. 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 genetic algorithm already used by [31] in a different context.

4.5 General Results

Table 1 summarizes the RMSE on MovieLens and Douban datasets. Confidence intervals correspond to a 95% range. 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. To the best of our knowledge, the best result published regarding MovieLens-10M (without side information) are reported by [18] and [3] with a final RMSE of respectively 0.7682 and 0.7769. However, those two methods require to recompute the full matrix for every new ratings. CFN has the key advantage to provide similar performance while being able to refine its prediction on the fly for new ratings. More generally, we are not aware of recent works that both manage to reach state of the art results while successfully integrating side information. For instance, [9, 12] reports a global RMSE above 0.8 on MovieLens-10M.

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 .

4.6 Impact of Side Information

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.

In order to exhibit the utility of side information, we report in Table 2 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 effective 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 0.13% and concatenating them increase the final score by a small margin

---

[3]The number of eigenvalues is arbitrary selected. We do not focus on optimizing the quality of this representation.
[4]Hyperparameters used for the experiments are provided with the source code.

Table 2: 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 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

of 0.20%. Therefore, V-CFN could handle the heterogeneity of side information. However, the U-CFN failed to use the friendship relationship to increase the RMSE.

5. REMARKS

5.1 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].

5.2 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. [27, 28] 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 3. At the time of writing, alternative strategies to train networks with sparse inputs on GPUs are under development.

6. CONCLUSION

---
[5]https://github.com/fstrub95/Autoencoders_cf

Table 3: 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	1,532MiB
MLens-20M	V	194M	34m45s	2,905MiB
MLens-1M	U	5M	7m17s	262MiB
MLens-10M	U	15M	34m51s	543MiB
MLens-20M	U	38M	59m35s	1,044Mib

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 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.

7. REFERENCES

[1] R. P. Adams and G. E. D. I. Murray. Incorporating side information in probabilistic matrix factorization with gaussian processes. In *Proc. of UAI'10*, 2010.

[2] C. M. Bishop. *Neural networks for pattern recognition*. Oxford univ. press, 1995.

[3] C. Chen, D. Li, Y. Zhao, Q. Lv, and L. Shang. Wemarec: Accurate and scalable recommendation through weighted and ensemble matrix approximation. In *Proc. of the Int. ACM SIGIR Conf. on Res. and Dev. in Information Retrieval*, pages 303–312, 2015.

[4] T. Chen, W. Zhang, Q. Lu, K. Chen, Z. Zheng, and Y. Yong. Svdfeature: a toolkit for feature-based collaborative filtering. *JMLR*, 13(1):3619–3622, 2012.

[5] G. Dziugaite and D. Roy. Neural network matrix factorization. *arXiv preprint arXiv:1511.06443*, 2015.

[6] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proc. of AISTATS'10*, pages 249–256, 2010.

[7] X. Glorot, A. Bordes, and Y. Bengio. Deep sparse rectifier neural networks. In *Proc. of AISTATS'11*, pages 315–323, 2011.

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

[9] Y.-D. Kim and S. Choi. Scalable variational bayesian matrix factorization with side information. In *Proc. of AISTATS'14, Reykjavik, Iceland*, 2014.

[10] Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009.

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

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

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

[14] Y. LeCun, L. Bottou, G. Orr, and K. Muller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–48. Springer, 1998.

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

[16] J. Lee, S. Kim, G. Lebanon, and Y. Singerm. Local low-rank matrix approximation. In *Proc. of ICML'13*, pages 82–90, 2013.

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

[18] D. Li, C. Chen, Q. Lv, J. Yan, L. Shang, and S. Chu. Low-rank matrix approximation with stability. In *Proc. of ICML'16*, 2016.

[19] S. Li, J. Kawale, and Y. Fu. Deep collaborative filtering via marginalized denoising auto-encoder. In *Proc. of CIKM'15*, pages 811–820. ACM, 2015.

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

[21] H. Ma, D. Zhou, C. Liu, M. R. Lyu, and I. King. Recommender systems with social regularization. In *Proc. of the int. conf. on Web search and data mining (WSDM '11)*, pages 287–296, 2011.

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

[23] A. Mnih and R. Salakhutdinov. Probabilistic matrix factorization. In *NIPS'07*, pages 1257–1264, 2007.

[24] I. Porteous and M. W. A. U. Asuncion. Bayesian matrix factorization with side information and dirichlet process mixtures. In *Proc. of AAAI'10*, 2010.

[25] S. Rendle. Factorization machines. In *Proc. of ICDM'10*, pages 995–1000, 2010.

[26] R. Salakhutdinov and A. Mnih. Bayesian probabilistic matrix factorization using markov chain monte carlo. In *Proc. of ICML'08*, pages 880–887. ACM, 2008.

[27] R. Salakhutdinov, A. Mnih, and G. Hinton. Restricted boltzmann machines for collaborative filtering. In *Proc. of ICML'07*, pages 791–798. ACM, 2007.

[28] S. Sedhain, A. K. Menon, S. Sanner, and L. Xie. Autorec: Autoencoders meet collaborative filtering. In *Proc. of Int. Conf. on World Wide Web Companion*, pages 111–112, 2015.

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

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

[31] O. Teytaud, S. Gelly, and J. Mary. Active learning in regression, with application to stochastic dynamic programming. In A. International Conference On Informatics in Control and Robotics, editors, *ICINCO and CAP*, pages 373–386, 2007.

[32] V. Tresp, S. Ahmad, and R. Neuneier. Training Neural Networks with Deficient Data. *Advances in Neural Information Processing Systems 6*, pages 128–135, 1994.

[33] T. T. Truyen, D. Phung, and S. Venkatesh. Ordinal boltzmann machines for collaborative filtering. In *Proc. of UAI'09*, pages 548–556. AUAI Press, 2009.

[34] A. Van den Oord, S. Dieleman, and B. Schrauwen. Deep content-based music recommendation. In *Proc. of NIPS'13*, pages 2643–2651, 2013.

[35] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P. Manzagol. Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion. *Jour. of Mach. Learn. Res.*, 11(3):3371–3408, 2010.

[36] H. Wang, N. Wang, and D.-Y. Yeung. Collaborative deep learning for recommender systems. *Proc. of Int. Conf. on Knowledge Discovery and Data Mining (KDD'14)*, 2014.

[37] X. Wang and Y. Wang. Improving content-based and hybrid music recommendation using deep learning. In *Proc. of the ACM Int. Conf. on Multimedia*, pages 627–636. ACM, 2014.

[38] Y. Wu, C. DuBois, A. Zheng, and M. Ester. Collaborative denoising auto-encoders for top-n recommender systems. In *Proc. of WSDM'16*, pages 153–162. ACM, 2016.

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

[40] F. Zhuang, D. Luo, X. Jin, H. Xiong, P. Luo, and Q. He. Representation learning via semi-supervised autoencoder for multi-task learning. In *Proc. of ICDM'15*, 2015.