Improvements on Deep Matrix factorization

Rafael Benatti, Clara Gard, Liora Taieb

Abstract

This report presents a detailed account of our work involving the implementation of a deep matrix factorization method, as described in Xue et al.'s work [1]. Our primary objective in this undertaking is to investigate the impact of experimental modifications on the loss function and the dataset. These modifications are aimed at facilitating data augmentation while also enhancing the accuracy and reducing the Root Mean Square Error (RMSE) of predictions in the context of recommendation systems. The focus of this report is rooted in empirical analysis and the quest for improved model performance. Throughout this report, we provide a comprehensive overview of the methodology employed, the specific alterations made, and the results obtained. The intent is to offer a transparent and informative account of the experimentation process, findings, and implications for the field of recommendation systems.

I Introduction

Initially, we implemented a regular matrix factorization which gave us a baseline for model comparison[1]. The matrix factorization task aims to represent a sparse matrix with a low rank matrix decomposed in two matrices.

$$R \approx IU^T \tag{1}$$

This is done in straight forward gradient descent optimization of the objective function minimizing the difference of R and IU^T. The objective is to minimize the mean square error between the observed ratings and the predicted ratings with two regularization terms, each one related to the norm of the matrices I and U.

$$\min_{I,U} \left\| R - IU^\top \right\|_F^2 + \lambda \left\| I \right\|_F^2 + \mu \left\| U \right\|_F^2 \tag{2}$$

The implementation of the matrix factorization resulted in the metric values is shown in Figure 1:

Metric	Value
Train RMSE	0.811
Test RMSE	0.882
Plate-form RMSE	0.930
Accuracy	23.79%

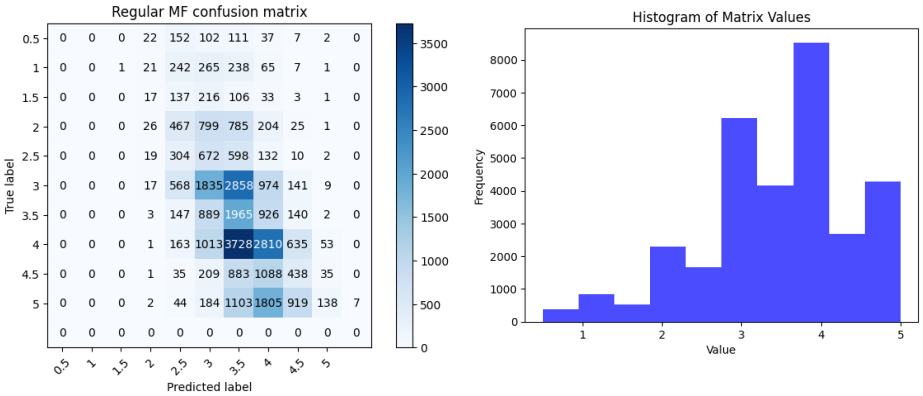


Figure 1: Regular MF metrics

As stipulated in the figure above, our baseline for the train RMSE is 0.811, and 0.882 for the test RMSE. We can firstly observe an increase in the metric when we passed the code in the plate-form : 0.930. We interpret this as Matrix Factorization over-fitting when the whole train and test sets are used to train the model.

If we look at the confusion matrix, we observe better predictions for non extreme values. Also, the values that were predicted the best are also the most present in our dataset, as we can see in our dataset distribution. The unbalance between values in the dataset and the absence of the lowest grades in the predictions motivated our experiments on data augmentation and regularization of the loss function.

[1]The code related to this approach can be found in the "MF.py" file

I.1 Evaluation of models

For the whole project, we use RMSE to estimate our models, as it is a relevant metric in our case. Moreover, we did not use the train and test sets we were given. We treated all the data as a single dataset that we would split ourselves each time the models were trained with ratio 0.8 for train and 0.2 for test. This was done to minimize over-fitting as much as possible.

II Deep Matrix Factorization

In all this section, we are referring to deep matrix factorization using 2 parallel networks for rows and columns of the matrix. The parameters were chosen with trials and errors and we finally settled on using 4 layers of size 610, 32, 16, 8 and 4980, 128, 64, 32 with both output size of 32 (this is the k of the matrix factorization, i.e. the number of features the neural network is extracting from the data). We are training this model using the BCELoss and the AdamW optimizer using weight decay = 0.0001.

The paper on which our deep matrix factorization is based uses all grades and a sample of non graded relations treated as zeros to compute the loss. While we initially tried this approach, it was giving poor results and the highest grade we could predict was around 3.5/5: the model was over-fitting on the zeros that were fed into the loss. To resolve this problem we tried using different weights for our loss function to lower the importance of the zeros. We used several instances of this to test which weights would fit best, and we found out that giving any value above 0 to the non-ratings would increase the loss of our test set. We suppose that it is better to ignore the zeros of the matrix because they are not actual ratings and this is what we try to predict. The use of zeros in the loss function might be useful for actual recommendation algorithms to not recommend movies the user is not likely to watch, but in the case of our problem, which is predicting grades accurately and not suggesting movies it is better to not take them into account.

	Train	Test	Plateform	Accuracy
MF	0.811	0.882	0.930	23.79%
DeepMF	0.814	0.865	0.852	24.93%

Figure 2: MF vs DeepMF metrics

With these metrics we can see that the deep matrix factorization performs better on the test set and is less susceptible to over-fit when adding more data as input (the results of MF were worse on the platform than on the test set).

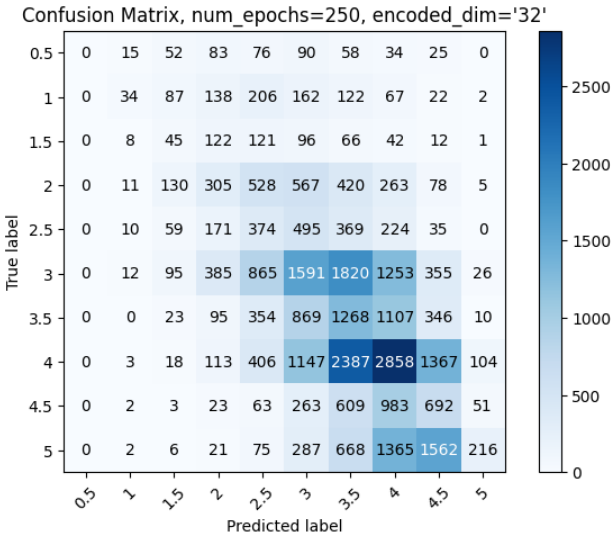
II.1 Loss function regularization

In the regular matrix factorization the objective function has two regularization terms, one for each matrix. The overall objective is to minimize the sum of these terms. Regularization helps in achieve a balance between fitting the train data and avoiding excessive complexity, which in some cases may also lead to poor generalization on unseen data. Similarly, we added two regularization terms in order to encourage our latent vectors to have small values.[2]
We can see small improvements in the model's generalization capacity, as we can see smaller differences in test and train RMSE when the regularization terms are used in the loss function. To have a great convergence, it demands a fine choice of hyper-parameters. With a great choice of hyper-parameters we were able to achieve RMSE and accuracy similar to the ones achieved with the regular deep MF. The model with regularization terms performed better in terms of prediction of lower grades.

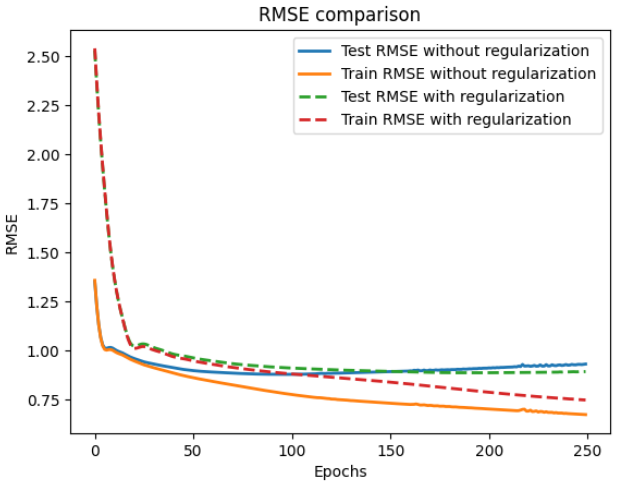
II.2 Chained Deep MF

Given the non-equalized distribution of classes in the dataset we also implemented an architecture with two chained deep matrix factorization networks. The first one would be trained with fewer epochs in order to make initial predictions to the model. Further, we would remove part of the high grades predictions and feed the new dataset with equalized classes into the network.

[2]The code related to this approach can be found in the "deepMF.py" file."



(a) Regularized model confusion matrix.



(b) RMSE comparison for both models with and without regularization terms.

Figure 3: Regular MF metrics

This is a really naive form of data augmentation and it didn't work to this model. In its best configuration, the same results of a deep MF network were produced. This happened because even the fewer epochs network was unable to produce reliable results with lower grades predictions. A sketch of the architecture can be seen in Figure 4.

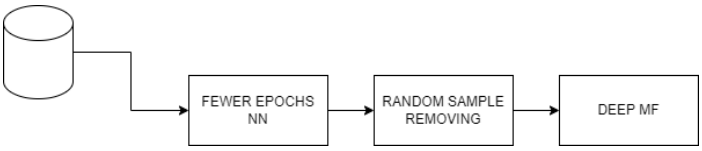


Figure 4: Chained Deep MF networks architecture

III Data augmentation

The training matrix contains $610 * 4980 = 3037800$ ratings, for which circa 3000000 are *nan*. This means only 1% of the matrix elements actually carry users preferences. The other 99% could be 2 things : a movie a user has not yet rated, or a movie a user doesn't want to rate. As we have seen in the previous section, this confusion negatively impacts the results and leads the research to go to the direction of data augmentation.

In this section, we try to make the most of the external knowledge we have on the movies - their genres. We will talk about 2 techniques we used to avoid the sparsity problem encountered in recommendation systems, to which our project is no exception. The first one link users' ratings to movie genres, and the second feed the neural network the genres of each movie, in hope it will learn new rating behaviors.

III.1 Adding ratings to the data

The naive approach to take into account external data is completing the training matrix. We tried to cluster the movies given their genres and release year, by assuming users are more drawn to same genres movies[3].

Remark. In literature, research has been made around the cold-start problem, which is the fact of having bad predictions for a new user with very few ratings. There is a nuance between this problem and the sparsity matrix problem. The technique we have implemented could offer some solution for cold-start, but only to a certain extent. Indeed, the more a user rated movies, the less random the new ratings will be.

[3]The code related to this approach can be found in the "data_completer.py" file.

Definition of a distance between 2 movies

To cluster movies together, we need to define a distance between them. We settled for the following metric d.

$$d(M_1, M_2) = 1 - I/U + |r_1 - r_2|/300$$

with I the common genres between M_1 and M_2, U the union, r_1 and r_2 the release year of M_1 and M_2, respectively.

This distance is 0 when 2 movies have the same genres and release year. It gets closer to 0 the more genres they have in common, and goes away from 0 when their release years are far apart. It satisfies what we are trying to achieve. Moreover, the re-normalization constant of the difference of release years has been chosen arbitrarily (there is 30 years between the oldest and newest movie of the database, we wanted the year to count less than the genres, hence the multiplication factor of 10).

Definition of a neighborhood of movies

Now that we defined the distance metric, we need to identify a set of neighbors for each movie, that is determine a range limit within it is relevant to characterize a movie as a neighbor of another one. Here is a set of information regarding this specific limit.

Neighbors per movie	**d $\leq$ 0.5**		**d $\leq$ 0.2**		**d = 0**	
	With year	W\out year	With year	W\out year	With year	W\out year
Mean	265	685	153	169	5	158
Max	796	1526	554	554	33	554
Number of added elements	2963854	2993799	1065841	1160729	88467	1054080

Table 1: Mean and max of neighbors and results on completed matrix given a distance d

As we can see, there is always a lot of neighbors in every case, even when the distance is 0. The year doesn't have much impact on the number of neighbors.

Results on deepMF

Metrics	**d = 0 w\out year**	**d $\leq$ 0 with year**
Train RMSE	0.71	0.78
Test RMSE	0.95	0.89

Table 2: Results of data augmentation on deepMF model, given a distance d setting the neighborhood

We kept only the completed matrices for which the number of added ratings was the lowest. As we experienced over-fitting with this method, the more we complete the initial matrix, the more the results will be satisfactory on the test set. For those added ratings, we put a weight of 0.3 to give them less importance, as they are assumed and not verified.

As expected, the model over-fits the data. This application decreases the train RMSE as we give it an easy rule to understand, but our assumption appears to be incorrect given the test RMSE. We have biased our model by applying to users a behavior they do not have, making the added external data inefficient.

III.2 One hot encoding of movie genres

To remove the bias added by directly importing new ratings, we tried to implement the genres of each movie directly into the model[4]. We simply added a matrix of size 19x4980 (19 genres for 4980 movies) to the left neural network composing deep matrix factorization (the one that extract features from the users).

What we observed were results very similar to those obtain by regular deepMF. A bias was not added, but the genres were not taken into account by the model. It would be interesting to extend this method to more external features, regarding movies or users.

[4]The code of this section can be found in "deepMF.py" and an example of it is written in "test.py"

IV Conclusion

To conclude, our attempts to improve deep matrix factorization for movie suggestions were not really successful. We learned that increasing the input data was not a good approach and that it leads to biases and sometimes over-fitting. One way we could try to improve the model would be taking into account genres and not seen movies to try and suggest movies that a user is susceptible to want to watch rather than purely basing the suggestions on ratings.
To go further, we could also improve the use of external data by insufflating meaning to the ratings. The paper [2] describes a semantic method using a PCA method that select the most significant external features available, and change the ratings.

References

[1] Hong-Jian Xue, Xin-Yu Dai, Jianbing Zhang, Shujian Huang, Jiajun Chen, *Deep Matrix Factorization Models for Recommender Systems*, Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence (IJCAI-17).

[2] Manuel Pozo, Raja Chiky and Elisabeth Metais, *Enhancing Collaborative Filtering using Implicit Relations in Data*, Institut Supérieur d'Eléctronique de Paris, LISITE Lab, 28, rue Notre-Dame-des-Champs. 75006 Paris, France.