Matrix Completion

Machine Learning and Big Data Processing

Daubry Wilson, Maes Cyprien, Szydelko Mateusz
https://github.com/cyprienmaes/Matrix_Completion_Algorithm
May 2020

***Abstract*—The aim of work summarized in this article is to perform Matrix Completion (MC) of a wide sparse matrix using the MovieLens data set. Several method are presented in the state of the art. A comparison is made between Collaborative Filtering (CF), AutoRec and Deep AutoRec (DAR), three different method used to perform matrix completion. Those methods are discussed and implemented using different architectures, then, experimental results are then provided and compared. The experimental results obtained shows that AutoRec achieves a good prediction on the MovieLens data set, deep auto-encoder achieves a better prediction but implies a more complexe Neural Network (NN). What comes out is that a good knowledge of the data set is important to choose the architecture of the NNs.**

***Index Terms*—AutoEncoders, Machine Learning, Matrix Completion, AutoRec, Deep Learning, Neural Network.**

I. Introduction

The goal of this project is to be able to recover a matrix from partial observations by completing the missing entries using the few entries observed. This can have multiple applications. For example, companies like Netflix try to predict their user's preference based on their favorite films to determine which film should be proposed to which user.

To perform this task, several methods can be considered. Either with or without using NN. Whatever the used technique, it is important to state that the MC problem is an ill-posed one. This means that for every input, there is not a single solution output but several possible solutions. Regardless of the method used, it is important to discuss the method used to check the "validity" of an output matrix. For the Netflix problem, a solution would be to compare new rates given by the users to the ones predicted by the algorithm.

This report is organized in four different sections. The first is a synthesis of the state of the art in MC. The second present the algorithms used in the implementation of MC. The third section compares different results obtains with the methods presented in the second section. The last section recapitulates and concludes this report.

In order to evaluate the quality of a MC system, several metrics are available. The one chosen in this project is the Root Mean Square Error (RMSE) of the prediction of the missing entries. The objective of the AutoEncoders used in this domain is solely to predict the data, the compression feature of those NNs is not the main objective. In that case, the RMSE of the training data would be used.

II. State of the art

Different methods achieving MC are presented in this section, which is split in two subsections. First, the collaborative filtering method is introduced. Second, the AutoEncoder (AE) are presented as well as the variants used to perform the MC. This section aims to give an understanding of the different methods that will be latter tested in the following sections.

A. Collaborative Filtering

One way to proceed to matrix completion is with CF. This does not use NN but rather matrix factorization. The idea behind this method is to reduce the big sparse matrix (n, m) with a rank smaller than the dimensions of the matrix to two matrix of size (n, r) and (r, m) where r is the rank of the sparse input matrix. The algorithm used in this project is explained in section III-A.

B. AutoEncoders

AE is an unsupervised deep learning method which is widely used for data reconstructions, compression and information retrieval. Those neural networks can also be used in other domains as image recognition, computer vision and speech recognition [7].

A basic AE is composed of three layers. An input one, a hidden one and an output layer, that architecture is displayed on Fig. 1[1], where n and m are the numbers of neurons of the layer, W and W' are two weights matrix of dimension (m, d) and (d, m) respectively.

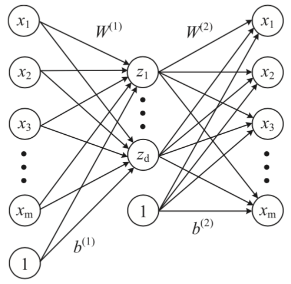


Figure 1: Architecture of a basic AE. Taken from [2]

[1]From left to right respectively

The encoder part of the network transform the input layer $x = \{x_1, ..., x_m\}$ into the hidden layer $z = \{z_1, ..., z_d\}$, with m being smaller than n following the eq. 1. s_f being an activation function and $b^{(1)} \in R^d$ a bias vector. Concerning the decoder, the eq. 2 follows the same idea, transforming the hidden layer into the output layer.

$$z = s_f(W^{(1)}x + b^{(1)}) \tag{1}$$

$$x = g_f(W^{(2)}z + b^{(2)}) \tag{2}$$

AE can be though as a compression and decompression system, the dimensions of the latent space being smaller than the dimension of the input/output layers. This network is trained by minimizing the reconstructions error between the input and output layer. One needs to use a sparse matrix as input with some missing values that should be known to compare the prediction obtained by the NN to the real values.

1) AutoEncoders for Matrix Completion: AutoRec and AutoEncoder based Matrix Completion (AEMC) are both AE respectively proposed by Fan et al. in [2] and Sedhain et al. in [6] designed to be used in MC. While in AE, only the parameters $W^{(1)}$ and $W^{(2)}$ are optimized to perform the best compression/reconstruction, in AEMC the missing entries recovery is also optimized simultaneously.

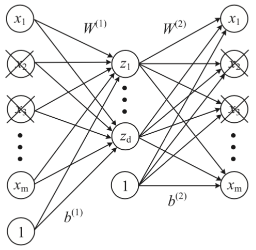


Figure 2: Architecture of an AEMC. Taken from [2]

The difference of this NN in comparison with an AE resides in the missing entries in the input and output layers. The loss function of an AE as in Fig. 1 can be written as in eq. 3. As said in the previous paragraph, the liberty degrees to minimize this function are $\{W^{(1)}, b^{(1)}, W^{(2)}, b^{(2)}\}$.

$$\begin{aligned} L = \frac{1}{2n}\sum_{i=1}^{n}(||x^i - g_f(W^{(2)}s_f(W^{(1)}x_i + b^{(1)}) \\ + b^{(2)}||^2) + \frac{\lambda}{2}(||W^{(1)}||_F^2 + ||W^{(2)}||_F^2) \end{aligned} \tag{3}$$

In the case of AEMC, the loss function remains mostly unchanged except for the presence of an Hadamard product between an Hadamard matrix and the content of the sum in eq. 3, this changed equation is shown at eq. 4. That Hadamard matrix is designed by Ψ, has the same dimensions as the input/output matrix and is only composed of ones and zeros. Zeros being located where entries are missing.

$$\begin{aligned} L = \frac{1}{2n}\sum_{i=1}^{n}\Psi_i \odot (||x^i - g_f(W^{(2)}s_f(W^{(1)}x_i + b^{(1)}) \\ + b^{(2)}||^2) + \frac{\lambda}{2}(||W^{(1)}||_F^2 + ||W^{(2)}||_F^2) \end{aligned} \tag{4}$$

The difference between AutoRec and AEMC resides in the objective function and the optimization method used. While AutoRec focuses on the same parameters as AE, AEMC also minimizes the loss functions using the same parameters as AE but also while focusing on the recovery of the entries at the same time : X: $\{W^{(1)}, b^{(1)}, W^{(2)}, b^{(2)}, X\}$..

In order to solve those problems, both network does not use the same method. AutoRec requires a gradient-based backpropagation, Sedhain et al. preferred Resilient backpropagation (RProp) to Light Broyden-Fletcher-Goldfarb-Shanno (L-BFGS) [6]. RProp improving the convergence speed and reducing the computational cost [5]. AEMC is solved using the nonlinear conjugate gradient which has also been preferred to L-BFGS for computational cost [2]. To use this algorithm efficiently, a preconditioning needs to be chosen.

It is important to note that the an AE can be used user- or film-based. Meaning that instead of reconstructing a matrix, a vector is actually reconstructed. One of the key being to conclude which one is the more optimized, either film- or user- based.

2) Deep AutoEncoders for Matrix Completion: Deep Learning Matrix Completion (DLMC) and DAR are NN which are respectively the evolution of AEMC and AutoRec. Those are proposed by Fan et al. in [2] and Kuchaiev et Ginsburg in [3]. The enhancement proposed rely on the hypothesis that having a multiple-hidden-layers NN is supposed to give better results than a single-hidden-layer NN by dealing with the non-linear part of the problem [1].

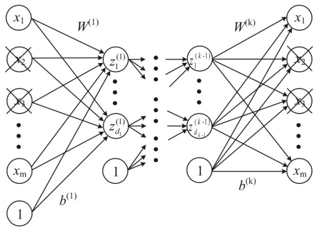


Figure 3: Architecture of an Deep AutoEncoders for Matrix Completion (DAEMC). Taken from [2]

The loss function in this case would be eq. 5 where $k \in 1, ..., j$, with j is the number of hidden layers plus one, this can be understood as a generalization of the eq. 4 to $(j-1)$ layers.

$$L = \frac{1}{2n}\sum_{i=1}^{n}\Psi_i \odot (||x^i - g^{(k)}(W^{(k)}(g^{(k-1)}(W^{(k-1)} \\ (...g^{(1)}(W^{(1)}x_i + b^{(1)})... + b^{(k-1)}) \\ + b^{(k)}||^2) + \frac{\lambda}{2}(||W^{(1)}||_F^2 + ||W^{(2)}||_F^2) \tag{5}$$

For DLMC, the optimization remains the same as for AEMC. For DAR, the Stochastic Gradient Descent (SGD) is used.

III. Algorithm and Architecture

This section describe the algorithms and architectures that are studied and experimented in section IV.

A. *Collaborative Filtering*

To perform the CF, we based our code on the third laboratory of the course Machine Learning and Big Data Processing (MLBDP) [4]. As said in section II-A, the sparse matrix R of dimension (n, m) is factorized in two matrices U and V of dimension (n, r) and (m, r) respectively, r being the rank of R. Ω is the set of the known entries and $\Omega|$ is the number of those known entries.

In order to compute those matrices U and V, a loss function is needed to minimize the Mean Square Error (MSE). This error is only computed for the value belonging to the set Ω. That loss function can be found in eq. 6. The two terms with the λ are the regularization terms needed to avoid over-fitting.

$$L = \frac{1}{2}\frac{\sum_{i,j\in\Omega}(U_i^T V_j - R_{ij})^2}{|\Omega|} + \frac{1}{2}\lambda\sum_{i,j\in\Omega}U_{ij}^2 + \frac{1}{2}\lambda\sum_{i,j\in\Omega}V_{ij}^2 \tag{6}$$

Since the problem needs to be solved for U and for V, we will use the Alternating least square (ALS) approach. Therefore, two different gradient descent will be used, one for U and another for V. Those gradient are the derivation of the eq. 6 by U and V respectively.

$$\frac{\partial L}{\partial U} = \sum_{i,j\in\Omega}(U_i^T V_j - R_{ij})V_j + \lambda U \tag{7}$$

$$\frac{\partial L}{\partial V} = \sum_{i,j\in\Omega}(U_i^T V_j - R_{ij})U_i + \lambda V \tag{8}$$

In order to test the validity of our prediction matrix, we duplicate the sparse input matrix and remove a fraction of the entries[2]. The diminished matrix is then used as the R matrix in the loss and gradient functions. To assess the validity of the prediction, we compare the removed entries with their values obtained from the prediction matrix.

[2]This depends on the implementation choices

B. *AutoEncoders algorithm and architecture*

The algorithm and architecture are implemented with the *tensorflow* and *keras* libraries. Both libraries are open-source and usable on *python*. They make the implementation of the architecture easier and create a very intuitive forward and backward propagation. The algorithm and implementation used in this project are close to [6] and [3] . Indeed, although AEMC and DLMC give very good results, backward propagation tools and conjugate gradient optimization are much more difficult to implement and are outside the scope of the project. Since minimization also takes the "to be completed matrix" as a parameter, backward propagation cannot be easily implemented and requires a thorough knowledge of the libraries available on *tensorflow* and *keras*.

1) AutoEncoders for Matrix Completion: The implementation of the AutoRec model is straight forward with the *tensorflow* and *keras* libraries. The loss function is described by eq. 5 where k = 2. For all AutoRec experiments λ (regularization strength) is set to 0.001. The choice is based on tests made with $\lambda \in \{0.001, 0.01, 0.1\}$ on MovieLens 100k. The results are shown in the appendix. Below, $g^{(2)}$ will be referred to as f and $g^{(1)}$ as g. RMSE(eq. 9) is used as metric in the proposed model. It is always fed with the training matrix followed by either the same training matrix or the validation data set. In the first case it shows the autoencoder's ability to reconstruct a known data set. In the second case it indicates the quality of the prediction of missing data. The prediction is clipped between 2 extremum which correspond to the minimum and maximum rate of the considered data set.

$$RMSE = \sqrt{\sum_{i=0}^{i=n}\frac{m_i(r_i - y_i)^2}{\sum_{i=0}^{i=n} m_i}} \tag{9}$$

The parameters in eq. 9 are the following one : m_i is the mask entry stating if the entry of the matrix need to be considered for the computation of the RMSE, r_i is the real know entry of the matrix and y_i is the predicted/reconstructed entry generated by the NN. A sum is made on every one of those value and is normalized based on the number of entries evaluated.

The developed program replicates loosely the one presented in [6]. In fact it takes advantage of the technical advances made in the MLBDP field since the publication of [6]. Instead of using R-prop, which is not suitable for big data sets, as optimizer for backpropagation, it uses Adam. The choice of Adam is supported by tests presented in the appendix. Results obtained using it are compared to the ones obtained for the same NN trained using RMS-prop, the last being related to R-prop.

The choice of a different optimizer impacts the behavior of the NN. Thus, the choice of activation functions and hyperparameters presented in [6] has to be reevaluated. The learning rate is set to 0.0001 for all the training and the mini-batch size is equal to 256.

2) Deep AutoEncoders for Matrix completion: The algorithm for DAR is pretty much the same as for AutoRec. The idea is to add hidden layers in the neural network considered. The weight and bias parameters are initialized by default using the glorot-uniform *keras* initializer. Each layer is fully interconnected with adjacent layers and forward propagation is applied throughout the entire network. Backward propagation is directly implemented by *keras* where several optimizations are available. Stochastic gradient descent is used in [3] as the default optimization. In the case of our algorithm, the Adam optimization is used on a mini-batch of 256. The loss function is determined by the equation 5. To avoid a very large number of comparison parameters, the activation functions and the regularizer are determined in advance and are taken directly from [3]. The activation functions between each layer are Scaled Exponential Linear Unit (SELU) already implemented in *keras* and the regularizer remains at 0.001. A layer is added after the latent. This one randomly dropping out some nodes of the latent layer to compute the decode layers. This dropout is used to avoid over-fitting in different architectures. As AutoRec already has this effect, it is a risk in DAR, so it is interesting to work with a dropout taking only 20% of the nodes as in [3]. The default parameters are summarized in the table I.

Hidden layer depth	3
Hidden layer size	256
Weight initializer	Glorot-uniform
Bias initializer	zeros
Weight regularizer λ	0.001
Activation functions	SELU
Optimizer & Learning rate	Adam (0.0001)
Batch size	256
Dropout	0.8

Table I: Default parameters of deep-AutoRec implemented on *keras*.

IV. Experimental results

In this section, the experimental results obtained using CF, AutoRec and DAR. For each NN, several architectures with several hidden layers, were tested on the MovieLens data set.

A. Collaborative Filtering

In order to assess the quality of the CF, the results are analyzed based on the Fig. 4, 5. The parameters used are the following one :

- $\texttt{iterations} = 250$
- $\alpha = 0.01$
- $\lambda = 0.0005$

What can be observed using Fig. 4 is that the cost function of the algorithm globally decreases with the number of iterations. When using 20 or 60 features, the cost function slightly increases at some points, before converging again. This can be explain by some divergence happening at some point, the algorithm being in a local minimum and finding another lower minimum of the cost function.

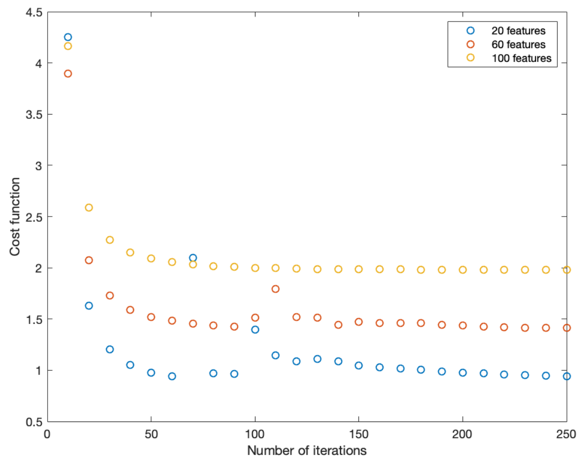


Figure 4: Cost function in function of the iteration

On Fig. 5, one can observe that the number of features influence the RMSE. Concerning the data train, it is clear that increasing the number of feature will lead to a better match between the factorization and the sparse matrix. Since more dimensions are available, the algorithm can reconstruct the data in a more loyal way, hence the better RMSE. Concerning the data prediction, the trend shows that this is diminishing up to a certain number of feature, 30 in this case, then a phenomena of over-fitting occurs. This happens when the factorization matches too well the data train and does not reconstruct the missing entries on the side.

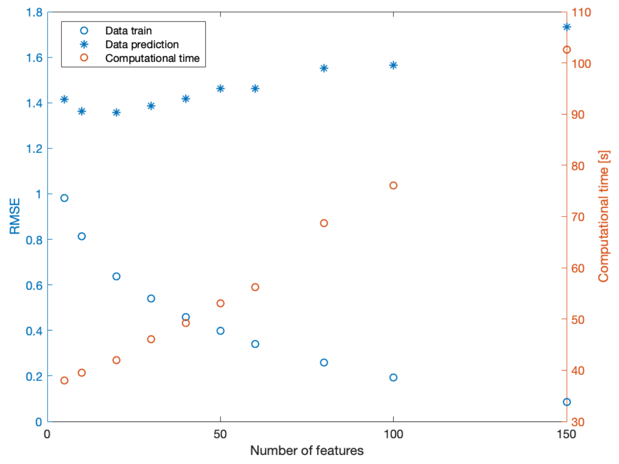


Figure 5: RMSE in function of the number of features

B. AutoRec

1) Latent's dimension: Figure (6) shows the validation RMSE for latent's dimensions $k = 64, 128, 256, 512, 1024$.

At $k = 512$ the slope of RMSE train is less stiff and the gain from doubling the latent size become marginal. The RMSE validation curve has a local minimum at that point. Thus 512 is used as the size if the latent for the rest of experiments. This results in accord with [6].

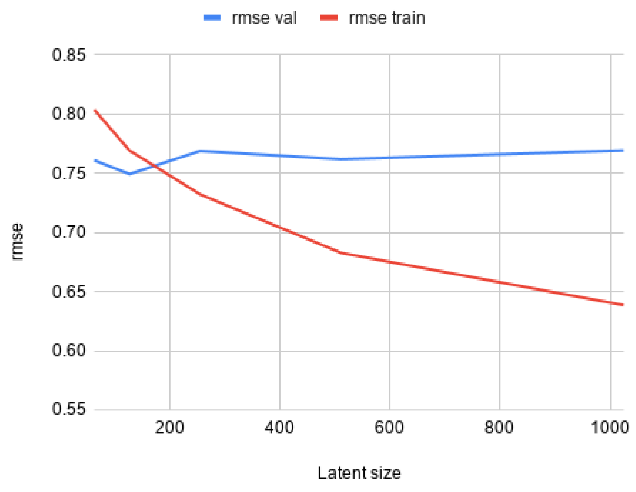


Figure 6: RMSE for different latent sizes.

2) Activation functions: Figures (7) and (8) show that the choice of the activation functions depends on how the data is preprocessed. In [6] it is proposed to replace the missing values by 3. As it shown on Figure (7b) most of the configurations will give the same results. The main difference between them is the intensity of the noisy behavior.

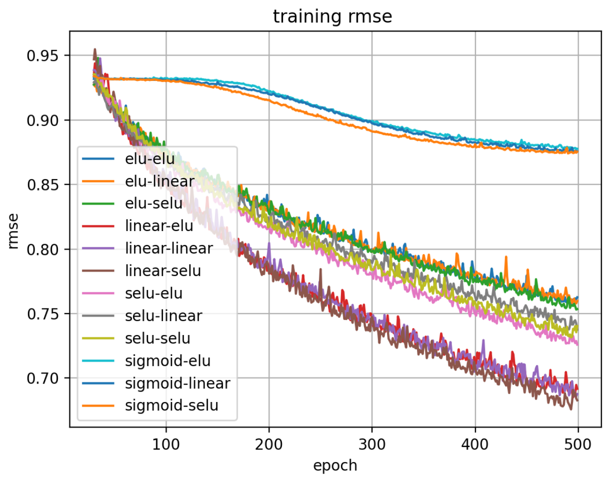
While the missing values are set to zero the NN is a much better autoencoder (Figure (8a)) however good predictions results will be obtained for fewer configurations (8b)). Only NN having sigmoid function as first activation deliver a good RMSE for validation.

$f(\cdot)$	$g(\cdot)$	RMSE train	RMSE validation	RMSE test
selu	elu	0.7534	0.7496	0.762
linear	selu	0.7375	**0.7455**	0.7578
linear	elu	0.7598	0.7507	0.760
elu	selu	0.7259	0.7510	0.761

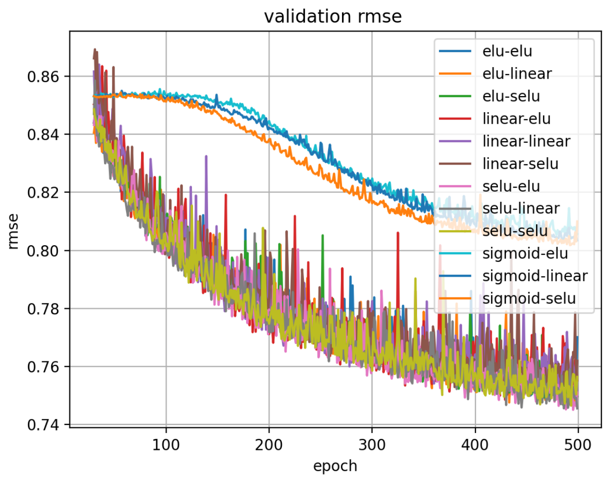
Table II: **AutoRec. Activation functions configurations with best validation RMSE.** NN configuration: k = 512, λ = 0.001, optimizer = Adam, learning rate = 0.0001, dataset = MovieLens 1m, input value for missing entries = 3.

$f(\cdot)$	$g(\cdot)$	RMSE train	RMSE validation	RMSE test
elu	sigmoid	0.7321	**0.7409**	**0.7462**
linear	sigmoid	0.7320	**0.7409**	**0.7460**
selu	sigmoid	0.7232	0.7419	0.7466
elu	selu	0.2937	0.91830	0.9270

Table III: **AutoRec. Activation functions configurations with best validation RMSE.** NN configuration: k = 512, λ = 0.001, optimizer = Adam, learning rate = 0.0001, dataset = MovieLens 1m, input value for missing entries = 0.



(a) Training data.

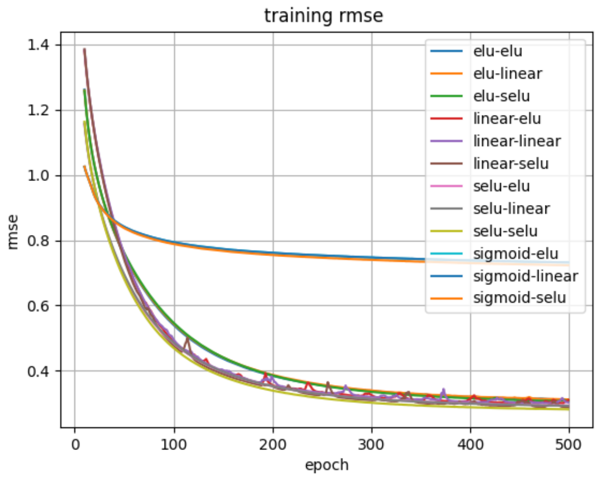


(b) Validation data.

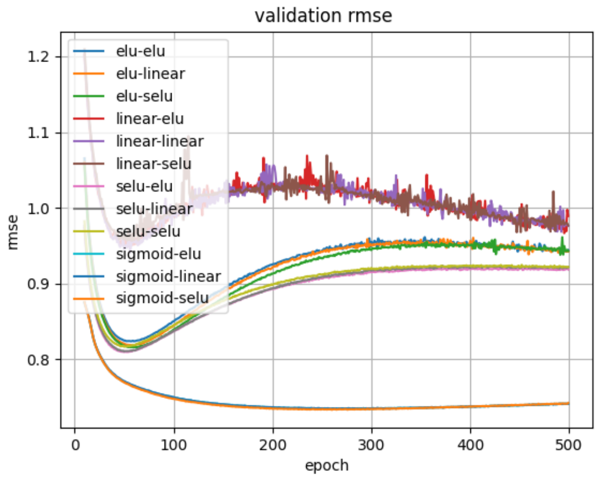
Figure 7: **RMSE for Autorec with different activation functions**. NN configuration: k = 512, λ = 0.001, optimizer = Adam, learning rate = 0.0001, dataset = MovieLens 1m, input value for missing entries = 3.

C. Deep AutoRec

1) RMSE comparison for different size in an AutoEncoder of three hidden layers: One of the first parameters to compare in "deep-autorec" is the size of each layer and of course the latent dimension. To do this, the architecture consists of 5 layers (one input, one output of the same size and 3 hidden layers). The sizes are chosen as in [3] and a symmetry between the encoder and the decoder is respected. The figure 9 shows the RMSE for the training and validation data. It is interesting to see that adding hidden layers improves the training and the higher the size of each layer, the better the training will be. Of course this criterion is not definitive and the validation RMSE shows the overfitting effect already visible in "AutoRec". This effect can be understood by the fact that by increasing the size

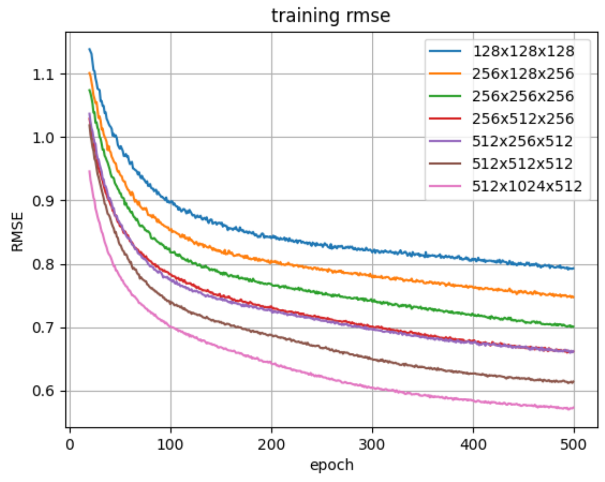


(a) Training data.

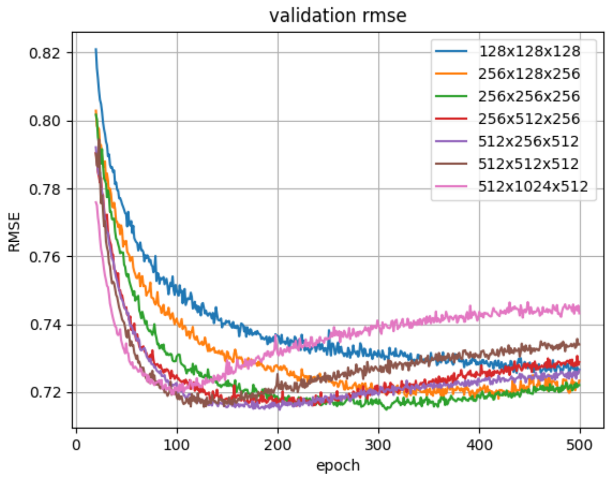


(b) Validation data.

Figure 8: **RMSE for Autorec with different activation functions**. NN configuration: k = 512, $\lambda = 0.001$, optimizer = Adam, learning rate = 0.0001, dataset = MovieLens 1m, input value for missing entries = 0.



(a) Training data.



(b) Validation data.

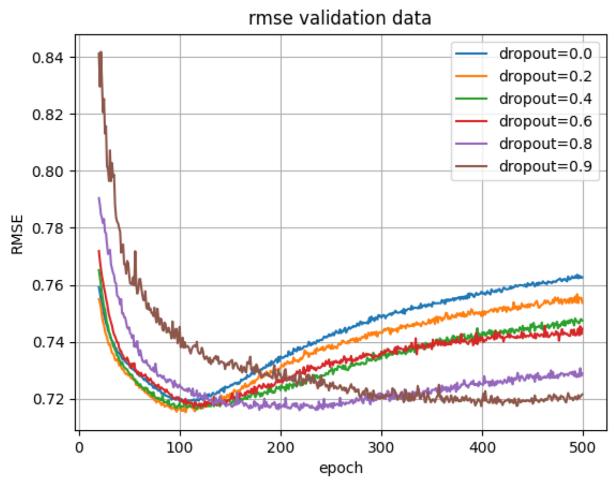
Figure 9: RMSE of deep-autoencoder of three hidden layers function of epochs. Adam optimizer with learning-rate of 0.0001, dropout of 0.8, weight regularizer of 0.001 and SELU activation functions.

of each hidden layer, more parameters, even with regularizer, will follow the trend of the training matrix. This effect is noticeable with the architecture 256x512x256, 512x256x512, 512x512x512, 512x1024x512. The dropout already planned in [3] and implemented here makes sense in these configurations.

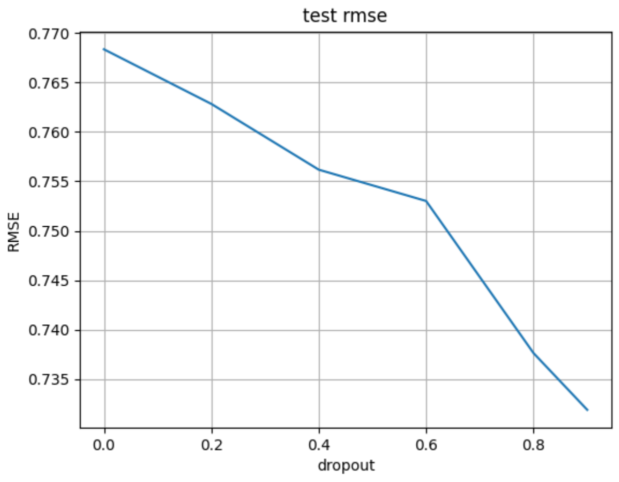
2) RMSE comparison for different dropout in a 256x512x256 AutoEncoder: The dropout, already explained in the algorithmic section, is an excellent way to avoid overfitting. Indeed, this parameter takes only a certain percentage of the nodes of the latent layer. The dropout is chosen between 0 included and 1 excluded (otherwise there is no more node). The results obtained for the RMSE validation and test data are presented in figure 10. In addition to canceling the overfitting, the RMSE of the test data also decreases as a function of the dropout. For example, for a 256x512x256 architecture, the RMSE of the data decreases to 0.73 without overfitting for a dropout of 0.9.

3) RMSE comparison for different hidden layer depth of size 256: The study of the size of a layer for a defined depth can be reversed. Figure 11 shows different depths for the same layer size of 256. We can see that the conclusion remains the same. When increasing the depth of the architecture, an overfitting appears and the RMSE obtained for the validation and test data increases. It is therefore important to properly determine the depth of the architecture or to determine other parameters (dropout, regularizer, etc.) to avoid this effect.

4) RMSE comparison for different learning rate of Adam optimizer of a 256x512x256 AutoEncoder: The learning rate

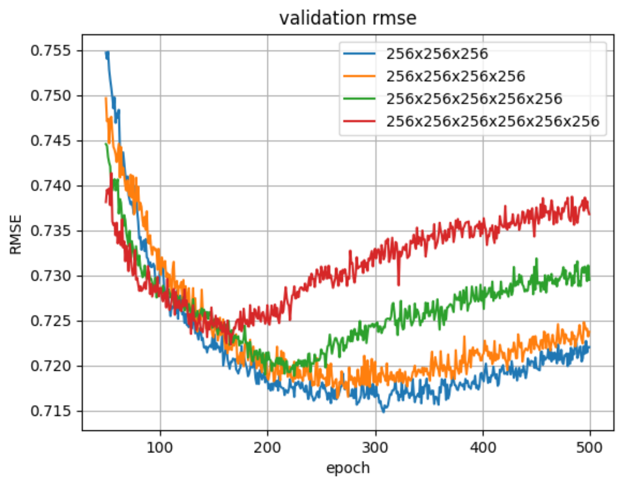


(a) Validation data.

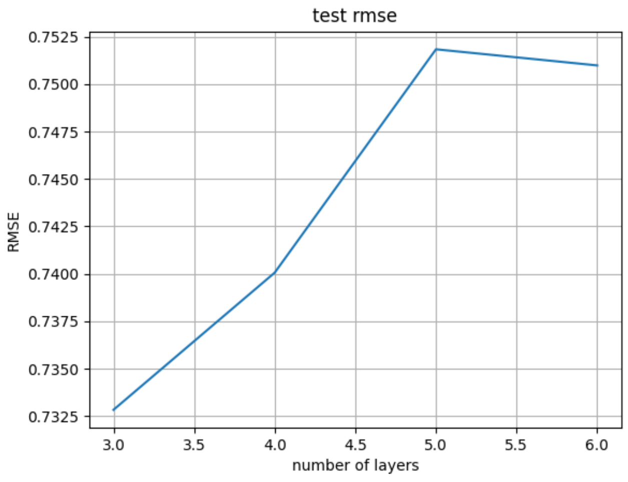


(b) Test data.

Figure 10: RMSE evaluation of deep-autoencoder 256x512x256 for validation and test data for different droupout. Adam optimizer with learning-rate of 0.0001, weight regularizer of 0.001 and SELU activation functions.



(a) Validation data.



(b) Test data.

Figure 11: RMSE evaluation for validation and test data of deep-autoencoder with different number of layers of size 256. Adam optimizer with learning-rate of 0.0001, dropout of 0.8, weight regularizer of 0.001 and SELU activation functions.

is a very important parameter in Adam optimization. Indeed, it describes the minimization step of the loss function. Thus, if the learning rate is too high, the optimization will never converge to a local minimum and if the learning rate is too low, the optimization may take a long time or be stuck in a local minimum higher than the desired one. Figure 12 shows several learning rates for a given neural network architecture (256x512x256). As we can see, a learning rate too high leads to a wrong estimation of the RMSE. The learning rate that seems to work best is 0.0001.

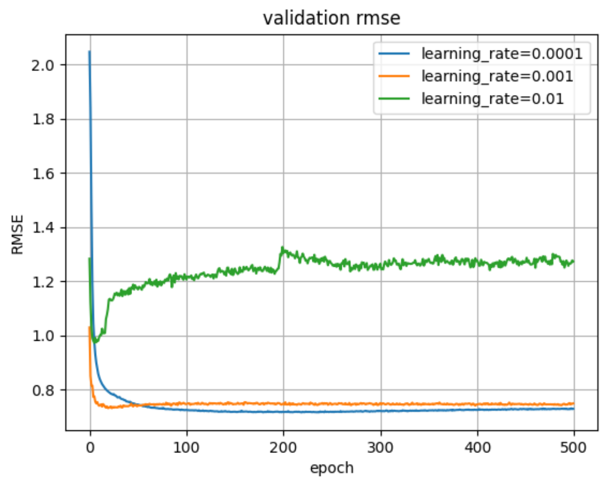
5) *Best result:* The table IV gives some of the best result for DAR.

architecture	256x512x256	256x(3)128x256	512x256x512x256
regularizer	0.001	0.001	0.001
optimizer	Adam	Adam	Adam
learning rate	0.0001	0.0001	0.0001
dropout	0.9	0.6	0.9
RMSE train	0.7251	0.6234	0.7019
RMSE validation	0.7223	0.7256	0.7248
RMSE test	0.734	0.7476	0.7419

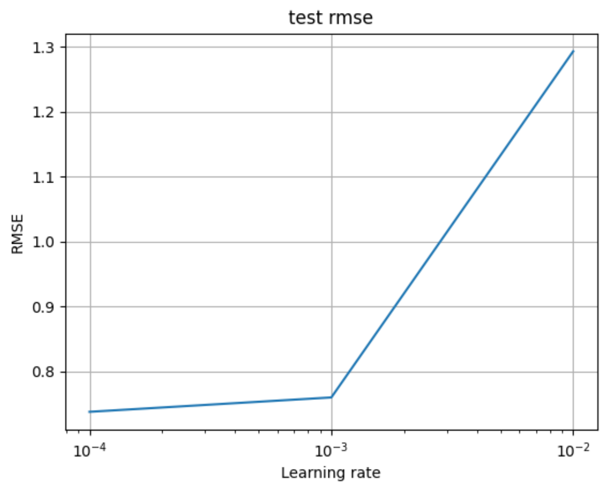
Table IV: Best result for DAR

V. Conclusion

In conclusion, several method achieving MC has been presented, implemented and tested. It directly appears that the Matrix Factorization (MF) method presents a rather bad result in comparison to AE NN. Concerning the AutoRec, based on



(a) Validation data.



(b) Test data.

Figure 12: RMSE evaluation for validation and test data of deep-autoencoder 256x512x256 for different learning rate. Adam optimizer, dropout of 0.8, weight regularizer of 0.001 and SELU activation functions.

Fig. 7b, 8 what can be concluded is that the choice of the initialization parameters of the matrix completely changes the response of the NN. The choices of the parameters will depend on those initialization parameters. Therefore, one can assume that a good knowledge of the data set is important to use an AE to perform MC. Using the tables II, III we arrive at the same conclusion, the data set will influence a lot the results of the NN. What can also be seen is that the best configurations in regard of the RMSE train will obtain the best results for the validation, one need to pay attention to over-fitting.

Concerning the DAR, increasing the size of each hidden layer will decrease the RMSE for the training but won't perform as good in term of validation, mostly due to an over-fitting. The NN being more focused on the reconstruction, performs less well in term of prediction. Concerning the dropout, Fig. 10 shows that increasing the value of the dropout will lead to a better MC. Increasing of depth of the NN will mostly lead to an over-fitting as it can be seen on Fig. 11b. In definitive increasing the number of hidden layers using a deep AE may lead to a better prediction of the missing entries but not as significantly as expected. However, a lot of different parameters can be tuned to create a really specific NN for each data set.

VI. Work division

The work was divided among the three members of the group. Each member worked on a specific technique, from the implementation to the test. Concerning the writing of the paper, the work was also divided, each member provided some references for the state of the art, which was written and corrected by each member. We were organizing a meeting on a weekly basis at the beginning, but we increased this frequency as the deadline was coming closer. During those meeting, each one of us presented what he has achieved since the last meeting, we brainstormed and shared ideas on the advancement of the project before stating the objectives for the following meeting.

References

[1] Pierre Baldi. Autoencoders, unsupervised learning, and deep architectures. In *Proceedings of ICML workshop on unsupervised and transfer learning*, pages 37–49, 2012.
[2] Jicong Fan and Tommy Chow. Deep learning based matrix completion. *Neurocomput.*, 266(C):540–549, November 2017.
[3] Oleksii Kuchaiev and Boris Ginsburg. Training deep autoencoders for collaborative filtering. *arXiv preprint arXiv:1708.01715*, 2017.
[4] Adrian Munteanu and Nikos Deligiannis. Course of : Machine learning and big data processing. VUB - Belgium, 2020.
[5] Martin Riedmiller and I. Rprop. Rprop - description and implementation details, 1994.
[6] Suvash Sedhain, Aditya Krishna Menon, Scott Sanner, and Lexing Xie. Autorec: Autoencoders meet collaborative filtering. In *Proceedings of the 24th international conference on World Wide Web*, pages 111–112, 2015.
[7] Guijuan Zhang, Yang Liu, and Xiaoning Jin. A survey of autoencoder-based recommender systems. *Frontiers of Computer Science*, 14(2):430–450, 2020.

Appendix A
AutoRec - Training Data

Dataset	Optimizer	Initial value for missing data	k	Lambda	f	g	Lr	Rmse train	Rmse val
MovieLens 1m	Adam	3	512	0.001	Linear	Linear	0.0001	0.6876	0.7514
				0.01				0.8588	0.7976
				0.001		elu		0.7598	0.7507
				0.01				0.8592	0.7849
				0.001	selu			0.7534	0.7496
				0.001	elu			0.7626	0.7701
				0.001	linear	selu		0.7375	0.7455
				0.01	linear			0.8610	0.7812
				0.001	selu			0.7403	0.7564
				0.001	elu			0.7259	0.7510
				0.001	elu	linear		0.6917	0.7552
				0.01				0.8603	0.7904
				0.001	selu			0.6739	0.7618
				0.01				0.8595	0.7852
				0.001	sigmoid			2.2997	2.0598
				0.01				2.2997	2.0598
				0.001	elu	sigmoid		0.8777	0.8056
				0.001	linear			0.8757	0.8034
				0.001	selu			0.8754	0.8100
		0		0.001	elu	elu		0.3114	0.9439
				0.001	linear			0.3121	0.9420
				0.001	selu			0.3067	0.9427
				0.001	elu	linear		0.3100	0.9892
				0.001	linear			0.3030	0.9761
				0.001	selu			0.2880	0.9764
				0.001	elu	selu		0.2937	0.9183
				0.001	linear			0.2927	0.9215
				0.001	selu			0.2816	0.9217
				0.001	elu	sigmoid		0.7321	0.7409
				0.001	linear			0.7320	0.7409
				0.001	selu			0.7232	0.7419
		3	64	0.001	selu	linear		0.8039	0.7610
			128	0.001				0.7693	0.7494
			256	0.001				0.7323	0.7689
			512	0.001				0.6827	0.7619
			1024	0.001				0.6386	0.7694
	RMSprop	3	512	0.001	Linear	Linear	0.0001		
				0.01					
				0.001		elu			
				0.01				0.8896	0.7969
				0.001		selu		0.8495	0.8046
				0.01				0.8998	0.8053
				0.001		sigmoid		0.8798	0.7919
				0.01				0.9765	0.8869

Dataset	Optimizer	Initial value for missing data	k	Lambda	f	g	Lr	Rmse train	Rmse val
MovieLens 100k	Adam	3	512	0.001	linear	Sigmoid	0.001	0.8467	
				0.01				0.9613	0.7215
				0.1				0.9981	0.7517
				0.001	Linear	Linear		0.3209	0.7044
				0.01				0.7774	0.6687
				0.1				0.9315	0.7024
				1				0.9670	0.7290
				0.001	elu	Linear		0.4612	0.6864
				0.01				0.8160	0.6636
				0.1				0.9350	0.7007
				0.001	Elu	elu		0.4733	0.6967
				0.01				0.8227	0.6701
				0.1				0.9524	0.6995
				0.001	linear	elu		0.4581	0.7230
				0.01				0.8156	0.6621
				0.1				0.9363	0.7026
				0.001	selu	selu		0.4059	0.7005
				0.01				0.8199	0.6663
				0.1				0.9389	0.7079
				0.001	selu	Linear		0.3972	0.7079
				0.01				0.8368	0.6724
				0.1				0.9434	0.7021