Neural collaborative filtering for network delay matrix completion

Sanaa Ghandi, Alexandre Reiffers-Masson, Sandrine Vaton, Thierry Chonavel

Neural collaborative filtering for network delay matrix completion

Sanaa Ghandi, Alexandre Reiffers-Masson, Sandrine Vaton, Thierry Chonavel
IMT Atlantique, LAB-STICC laboratory
Brest, France
{sanaa.ghandi, alexandre.reiffers-masson, sandrine.vaton, thierry.chonavel}@imt-atlantique.fr

***Abstract*—In network monitoring, delays are of great use when it comes to QoS or content distributed services. However, it is often impossible to have access to all the delay measurements within a network. This can be due to network failures or to established measurement policies. For these reasons, delay matrix completion techniques are important for an optimal network monitoring service. In this paper, we formulate the completion problem as a neural collaborative filtering problem by testing two different architectures, generalized matrix factorization and multi-layer perceptron. We evaluate these methods on two different datasets: a synthetic one generated by an autonomous system simulator, and a real-world dataset from Ripe Atlas platform. Finally, a comparative study is conducted between these neural collaborative filtering methods and standard approaches.**

Keywords: Internet delays, matrix completion, deep learning

I. Introduction

In recent years, matrix completion has gained a lot of popularity with applications ranging from recommender systems, image and audio reconstruction, to networking and genomics [1]. Until recently, the matrix completion problem was resolved using matrix factorization approaches [2]. Lately, other non-linear methods have appeared to solve this problem, such as neural networks. These approaches can be more interesting in terms of computational cost and their ability to capture the complexity of high dimensional data [3].

In this work, we consider a matrix completion problem in the context of Internet delays. When network data are studied, we are often confronted with problems of missing data, either because we cannot continuously probe the network or because certain equipment do not respond. This may typically concern a set of source and destination nodes and the delays between them, such as the Round Trip Time (RTT). This work addresses the delay matrix completion problem from a neural network perspective by using a neural collaborative filtering (NCF) approach [3]. Collaborative filtering [4] is a widely used technique in recommender systems. It gathers data from other users in order to identify similarities between them. This helps the recommender system to predict missing users preferences. Neural collaborative filtering in particular, makes use of the flexibility and complexity of neural networks to accomplish this recommendation task. We compare the performance obtained with other techniques based on non-negative matrix factorization (NMF) [5]. We base our comparison on two datasets. The first generated by a delay simulator of an autonomous system with variations linked to external routing changes [6]. The second one consists of real RTT measurements on the Internet obtained from the RIPE Atlas platform [7].

In Section II we present our methodology consisting of two neural collaborative filtering architectures. In Section III we evaluate their performance by conducting extensive experiments and we compare them to a previous work based on some standard matrix factorization techniques. Finally, in section IV we present our conclusion and some future work directions.

II. Methodology

In this paper, we address the estimation of the missing delays in a network as a matrix completion problem. For this purpose, we consider a network with different nodes that can refer to anchors or probes, and we consider slotted time. We suppose that we can run delay measurements in the network, at every instant, between a pair of nodes: one considered a *source* and the other a *destination*. In general, these measurements are conducted every 5 minutes in order to monitor properly the network state.

More precisely, let $\boldsymbol{D}$ denote our $n \times m$ delay matrix, where n is the number of *source-destination* pairs in the network and m the number of instants. In this paper, we assume that there are some *unobserved entries* in the matrix $\boldsymbol{D}$ that correspond to missing measurements. The problem is to find these missing values by exploiting the spatial and temporal dependency between the different delays.

Let us introduce $\mathbb{S}$ the set of index pairs (i, j) such that $\boldsymbol{D}_{ij}$ is *observed*, and $\overline{\mathbb{S}}$ the set of indices of *unobserved* delays. Each index (i, j) of an entry of the delay matrix can be represented by two one-hot encoded vectors. For example, we associate to the entry $\boldsymbol{D}_{ij}$ with $(i, j) \in \mathbb{S}$ the vectors $\boldsymbol{p}_i \in \{0, 1\}^n$ and $\boldsymbol{t}_j \in \{0, 1\}^m$ where:

$$\boldsymbol{p}_i(k) = \mathbb{1}_{\{k=i\}} \quad \text{and} \quad \boldsymbol{t}_j(l) = \mathbb{1}_{\{l=j\}} \tag{1}$$

where $\mathbb{1}$ is the indicator function. An example of this encoding is given in Fig. 1.

We are now going to present two different Neural Collaborative Filtering architectures that can be used to solve this matrix completion problem. The Generalized Matrix Factorization (GMF) which generalized the standard matrix factorization

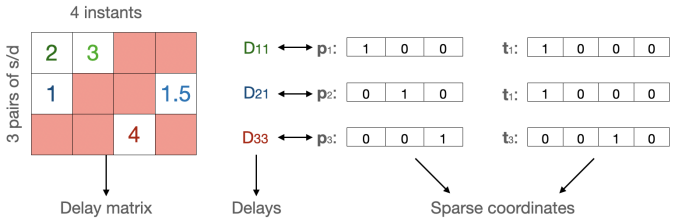

Fig. 1. Sparse representation of the delays coordinates, pink squares represent the missing delays.

approach to learn the model and the multi layer perceptron (MLP), a standard deep learning architecture [3].

Each architecture is trained on a given number of epochs over the set $\mathbb{S}$ of observed delays in order to minimize a loss, defined as a measure of discrepancy between the predicted and the observed delays. In order to do so, the neural network uses an optimizer that updates each parameter during the back propagation with a given learning rate. The impact of all these parameters on the quality of the reconstruction will be studied in the evaluation section. On the other hand, the test phase is performed on $\overline{\mathbb{S}}$. To be more precise, we act as if the values were not known, and we check the quality of the prediction by evaluating the loss between the actual and the predicted values.

A. Generalized Matrix Factorization (GMF)

According to [3], the architecture of the GMF algorithm is composed of three main components.

1) For each $\boldsymbol{D}_{ij}$ to be predicted, the *input layer* consists of sparse entry vectors $\boldsymbol{p}_i$ and $\boldsymbol{t}_j$.
2) The *embedding layer* is a fully connected layer projecting a sparse vector to a dense one. Its dimension k is considered as a parameter of the model. This layer is seen as a latent feature extractor.
3) Finally, the *neural collaborative filtering block* is a multi-layered neural architecture that connects the output of the embedding layer to the predicted delays. Its goal is to minimize the loss between the estimated delay and the target value.

In this model, each layer output serves as an input for the next one. This architecture is described in figure 2, with the size indicating the dimension of the input, with k, r and 1 corresponding respectively to the dimension of the embedding layer, the FC_1 and the FC_{out} with FC denoting a fully connected layer. The ReLu is the activation function used between the two FCs. We can formulate the GMF model as:

$$\begin{aligned}&\boldsymbol{D}^e_{ij}(\boldsymbol{\Xi_1}, \boldsymbol{\Xi_2}, \boldsymbol{\Theta}_1, \boldsymbol{\Theta}_{out})\\ &= \phi_{out}(\phi_1(E_1(\boldsymbol{p}_i, \boldsymbol{\Xi_1}) \circ E_2(\boldsymbol{t}_j, \boldsymbol{\Xi_2}), \boldsymbol{\Theta}_1), \boldsymbol{\Theta}_{out}),\end{aligned} \tag{2}$$

where $\boldsymbol{D}^e_{ij}$ is the estimated delay corresponding to the entry (i, j), ϕ_{out} and ϕ_1 are respectively the mapping functions of the output layer FC_{out} and the first neural collaborative filtering layer FC_1. $\circ$ denotes the element-wise product of vectors

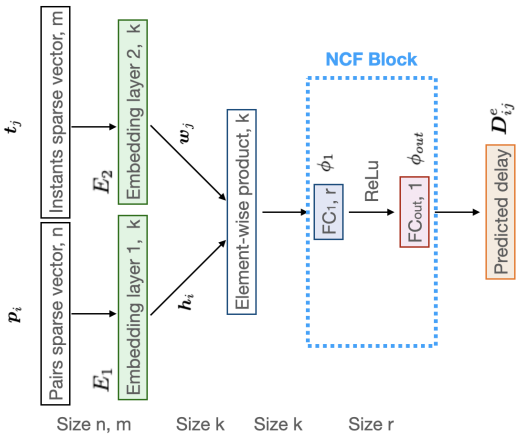

Fig. 2. Generalized Matrix Factorization (GMF) model.

and E_1 and E_2 refer to the functions of the embedding layers. Finally, $\boldsymbol{\Theta}_1$, $\boldsymbol{\Theta}_{out}$, $\boldsymbol{\Xi_1}$ and $\boldsymbol{\Xi_2}$ correspond to their trainable parameters. For simplicity reasons, $\boldsymbol{D}^e_{ij}(\boldsymbol{\Xi_1}, \boldsymbol{\Xi_2}, \boldsymbol{\Theta}_1, \boldsymbol{\Theta}_{out})$ is denoted by $\boldsymbol{D}^e_{ij}(\boldsymbol{\Xi}, \boldsymbol{\Theta})$.

Such neural network is called the Generalized Matrix Factorization (GMF) since we can find the classical matrix factorization if ϕ_1 is a product by the all one vector and ϕ_{out} is the identity function. Indeed, let us denote $\boldsymbol{h}_i = E_1(\boldsymbol{p}_i, \boldsymbol{\Xi_1}) \in \mathbb{R}^k$ and $\boldsymbol{w}_j = E_2(\boldsymbol{t}_j, \boldsymbol{\Xi_2}) \in \mathbb{R}^k$ the dense vectors resulting from the embedding layers. The classical reduced rank matrix factorization model $\boldsymbol{D} = \boldsymbol{H}^\top \boldsymbol{W}$ with $\boldsymbol{H} = [\boldsymbol{h}_1 \cdots \boldsymbol{h}_n] \in \mathbb{R}^{n \times k}$ and $\boldsymbol{W} = [\boldsymbol{w}_1 \cdots \boldsymbol{w}_m] \in \mathbb{R}^{k \times m}$ estimates $\boldsymbol{D}^e_{ij}(\boldsymbol{\Xi}, \boldsymbol{\Theta})$ as follows: $\boldsymbol{D}^e_{ij}(\boldsymbol{\Xi}, \boldsymbol{\Theta}) = \boldsymbol{h}_i^\top \boldsymbol{w}_j$. When $\phi_{out} = I_d$ and $\phi_2 = \mathbf{1}$ with I_d the identity function and $\mathbf{1}$ a vector of ones of length k, and there is no intermediate ReLu:

$$\begin{aligned}&\boldsymbol{D}^e_{ij}(\boldsymbol{\Xi}, \boldsymbol{\Theta}) = \phi_{out}(\phi_1(\boldsymbol{h}_i \circ \boldsymbol{w}_j, \boldsymbol{\Theta}_1), \boldsymbol{\Theta}_{out})\\ &= I_d(\mathbf{1}^\top(\boldsymbol{h}_i \circ \boldsymbol{w}_j)) = \mathbf{1}^\top(\boldsymbol{h}_i \circ \boldsymbol{w}_j) = \boldsymbol{h}_i^\top \boldsymbol{w}_j.\end{aligned} \tag{3}$$

B. Multi layer perceptron (MLP)

The MLP model also takes as an input the sparse vectors $\boldsymbol{p}_i$ and $\boldsymbol{t}_j$ followed by an embedding layer. The embedded vectors $\boldsymbol{h}_i$ and $\boldsymbol{w}_j$ are then concatenated and provided to a towered multi-layered architecture. This multi-layer architecture introduces more flexibility and non-linearity to the model. We denote $\phi_i(\boldsymbol{x}, \boldsymbol{\Theta}_i)$ the mapping function of each hidden layer i and $\boldsymbol{\Theta}_i$ its trainable parameter. For this model, we choose the ReLu as an activation function.

C. Loss and optimizers

For both architectures, we consider the mean squared error as the loss function:

$$L(\boldsymbol{\Xi}, \boldsymbol{\Theta}) = \frac{1}{|\mathbb{S}|} \sum_{(i,j) \in \mathbb{S}} (\boldsymbol{D}_{ij} - \boldsymbol{D}^e_{ij}(\boldsymbol{\Xi}, \boldsymbol{\Theta}))^2, \tag{4}$$

with $|\mathbb{S}|$ is the cardinal of $\mathbb{S}$ and $\boldsymbol{\Xi}, \boldsymbol{\Theta}$ denoting the different trainable parameters of the network. In the evaluation section, we use a batch size of 1 and we test different optimizers.

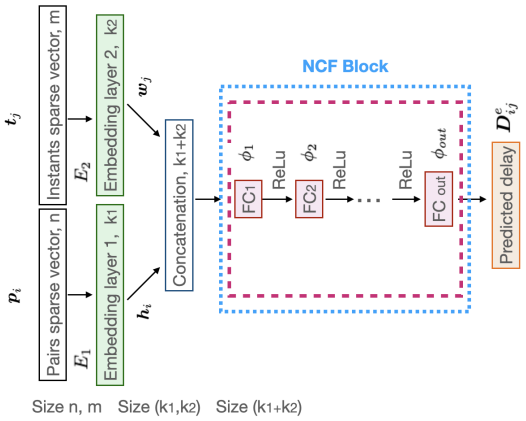

Fig. 3. Multi Layer Perceptron (MLP) model.

III. Evaluation

We evaluate these two methods on two different datasets. The first one is a synthetic dataset generated by an autonomous system (AS) simulator [5], [6]. This simulator generates the traffic entering and leaving the AS, with some abrupt traffic changes that can be interpreted as BGP routing changes. The distribution of the traffic on the links of the AS network is calculated using a simple gravity model. Finally, the delays on each link are modeled through an M/M/1 model. In this simulator, the internal routing is supposed to be known and fixed. The second dataset is a real-world dataset from a measurement campaign that we have conducted on the Ripe Atlas platform [7]. This campaign involves Ripe anchors located around the world and the measurements are conducted at a frequency of 4 minutes.

To evaluate the completion performance of the two NCF architectures, we consider the convergence stress [8] as a performance measure. It measures the quality of the reconstruction on the missing values $\boldsymbol{D}_{ij}$ with $(i,j) \in \overline{\mathbb{S}}$:

$$Stress = \sqrt{\frac{\sum_{(i,j)\in\overline{\mathbb{S}}}(\boldsymbol{D}_{ij} - \boldsymbol{D}^e_{ij}(\boldsymbol{\Xi},\boldsymbol{\Theta}))^2}{\sum_{(i,j)\in\overline{\mathbb{S}}}(\boldsymbol{D}_{ij})^2}} \quad (5)$$

We use the stress to compare the performance of NCF approaches to the Non-negative Matrix Factorization (NMF) ones [5]. To this end, we consider two NMF algorithms: the alternated projected gradient (APG) [9] and the NeNMF [10] algorithm that uses a Nesterov gradient. NMF is used with a view to highlight clusters of delay trajectories in the lines of the right matrix of the factorization, hence the positivity constraint for the factorization.

A. Results on synthetic data

Impact of the embedding layer dimension Our simulated dataset consists of a delay matrix of $n = 20$ *source-destination* pairs over $m = 400$ instants. We start our study by varying the embedding layer dimension k in the model architectures.

As we can see in Table I the stress decreases with the embedding layer dimension. This is due to the fact that a larger embedding layer captures more latent features and is more adapted to complex high-dimensional data. We set the embedding dimension to 35 for the MLP approach since it reaches its minimum stress at this value, whereas for the GMF the embedding dimension will be fixed at 30.

Impact of the number of epochs We have also assessed the influence of the number of epochs on the completion accuracy by changing it during the training phase. We can notice in figure 4 (a) that the performance is enhanced when we increase the training phase for both GMF and MLP. We can see that stress starts to become stable between 1.5e5 and 2e5 epochs. We fix the number of epochs to 1.5e5 for the rest of the experiences for both architectures.

Impact of the optimizer and the learning rate We are also interested in the impact of the type of optimizer. We have tried three different optimizers: the stochastic gradient descent (SGD), the Adam optimizer and the AdamW which corresponds to the Adam optimizer with weight decay. Table I shows that the Adam optimizer is optimal for this study for both models. Moreover, we analyzed the influence of the learning rate by trying different values. Figure 4 (b) indicates that the learning rates of 0.001 and 0.0005 minimize the stress respectively for the GMF and MLP models. Hence, the learning rate will be fixed for each architecture accordingly.

Comparison of the execution times The execution time plays a crucial role in the real-world deployment of completion methods and should take part in the evaluation process. By using a $2,6$ GHz Intel Core i7 processor and a 32Go 2667 MHz DDR4 memory the MLP and GMF were executed respectively within 335 and 222 seconds each. On the other hand, NeNMF takes 472s to converge whereas APG has a much longer execution time of 16 minutes.

Testing piecewise regularization term on NCF As previously highlighted in [5], we can add a regularization term to the loss in order to incorporate more information about the temporal stability of the RTTs observed in the data. Let us denote by $L_\beta(\boldsymbol{\Xi},\boldsymbol{\Theta})$ the regularized loss function, where β is the regularization parameter. Omitting the dependencies with respect to $(\boldsymbol{\Xi},\boldsymbol{\Theta})$ for simplicity reasons, L_β writes:

$$\begin{aligned} L_\beta = \; & \tfrac{1}{|\mathbb{S}|}\textstyle\sum_{(i,j)\in\mathbb{S}}(\boldsymbol{D}_{ij} - \boldsymbol{D}^e_{ij})^2 \\ & +\beta\textstyle\sum_i\big(|\boldsymbol{D}^e_{i1} - \boldsymbol{D}^e_{i2}| + |\boldsymbol{D}^e_{i(T-1)} - \boldsymbol{D}^e_{iT}| \\ & +\textstyle\sum_{t=2}^{T-1}|\boldsymbol{D}^e_{it} - \boldsymbol{D}^e_{i(t+1)}| + |\boldsymbol{D}^e_{it} - \boldsymbol{D}^e_{i(t-1)}|\big), \end{aligned} \quad (6)$$

We observe in table III that the regularization does not have a positive impact on the stress evolution. Therefore, we will fix $\beta = 0$.

Impact of the sampling rate and NMF/NCF comparison When a signal is partially observed, we denote the sampling rate, the proportion of the observed values with respect to the total entries of the matrix. In figure 4 (c), we investigate its impact on the stress for NCF and NMF methods. Despite the fact that all the methods display a stress less than 2%, we can

Emb dim k	5	10	15	20	25	30	35
Stress GMF	0.0183	0.0188	0.0134	0.0133	0.012	**0.0103**	0.0106
Stress MLP	0.0222	0.0135	0.0154	0.0113	0.0144	0.0108	**0.0085**

TABLE I

IMPACT OF THE EMBEDDING DIMENSION ON THE STRESS.

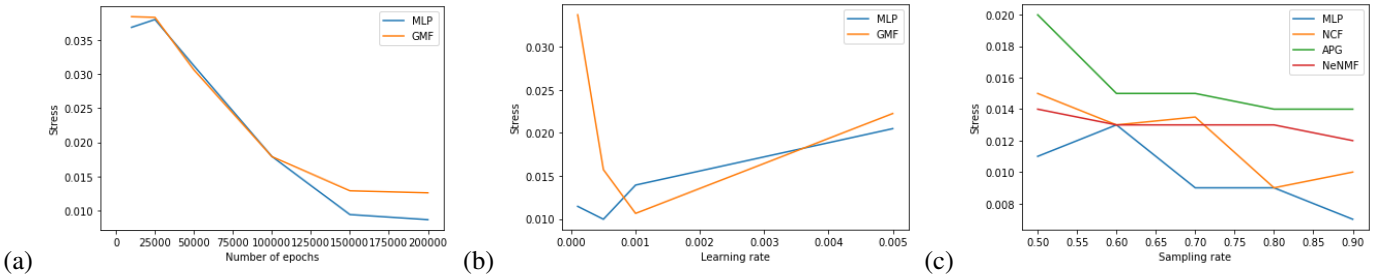

Fig. 4. (a) The impact of the number of epochs, (b) the impact of the learning rate and (c) the impact of the sampling rate on the stress.

notice that the MLP clearly outperforms the other methods with a stress smaller than 1% for sampling rates higher than 70%. On the other hand, the performance of the GMF tends to approach the NeNMF results on lower sampling rates but surpasses it on higher ones. APG displays however worse results than other methods.

Figures 5 and 6 show the reconstruction of two delay matrices sampled respectively at 70% and 50% using GMF and MLP. We can notice that the reconstruction captures the baseline of each time series. Besides, the completion corresponding to a higher sampling rate is less noisy and shows more stability.

Optimizer	SGD	Adam	AdamW
Stress GMF	0.0194	**0.0112**	0.0210
Stress MLP	0.0344	**0.0089**	0.0102

TABLE II

IMPACT OF THE OPTIMIZER ON THE STRESS.

Beta	0	0.01	0.1	0.3	0.5
Stress GMF	**0.0104**	0.0130	0.0201	0.0230	0.0260

TABLE III

IMPACT OF BETA ON THE STRESS.

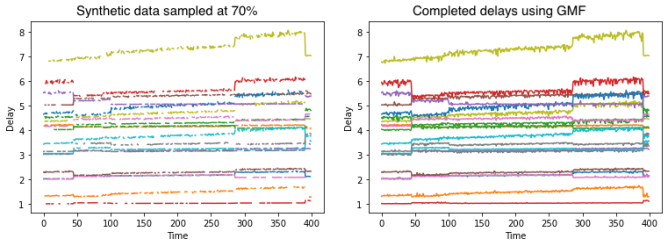

Fig. 5. Reconstruction of an 70% observed simulated dataset using GMF

B. Results on real-world data

This dataset comes from a measurement campaign on Ripe Atlas. Unlike the precedent work [5], we wanted to have a ground truth for all the entries of our delay matrix. To this end, we selected anchors belonging to some of Google ASs around the world. They are prone to be more stable, and have a higher probability to be connected and respond to requests. This was actually the case, since we did not have missing delays on this campaign. The dataset contains 50 RTT time series of length 800 each corresponding to 22 hours of measurements.

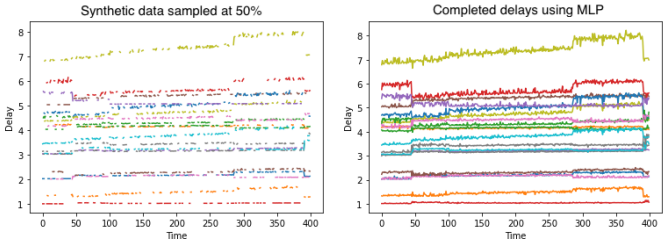

Fig. 6. Reconstruction of an 50% observed simulated dataset using MLP

Impact of the embedding layer dimension First, we explore the impact of the embedding layer on the stress in table IV. These values were achieved by running 1.5e5 epochs. The stress remains overall constant for both architectures, but we can see that GMF and MLP achieve their minimum stress at 20. For this reason, we fix this dimension to 20.

Emb dim	5	10	15	20	25	30
Stress GMF	0.0118	0.0115	0.0115	**0.0113**	0.0115	0.0118
Stress MLP	0.0121	0.0134	0.0132	**0.0119**	0.0142	0.0131

TABLE IV

IMPACT OF THE EMBEDDING DIMENSION ON THE STRESS FOR 1.5e5 EPOCHS.

Impact of the number of epochs Table V shows that MLP reaches its minimum stress at 1.5e5 epochs, whereas GMF stress continues to decrease after this value. Due to time execution considerations, we set the number of epochs to 1.5e5 for both architectures.

Impact of the optimizer and the learning rate We investigate the impact of the learning rate at Table VI. One can clearly see that 1e−4 is the best learning rate for both architectures.

Number of epochs	5e4	1e5	1.5e5	2e5	2.5e5
Stress GMF	0.0182	0.0117	0.0119	0.0116	**0.0115**
Stress MLP	0.0123	0.0120	**0.0118**	0.0119	0.0142

TABLE V

IMPACT OF THE NUMBER OF EPOCHS ON THE STRESS.

We fix then the learning rate at this value. Table VII shows the stress after $1.5e5$ epochs with a learning rate of $1e{-}4$ for different optimizers. The Adam optimizer is the most suitable for both the GMF and the MLP architectures.

Optimizer	SGD	Adam	AdamW
Stress GMF	0.4433	**0.0113**	0.0116
Stress MLP	0.4408	**0.0117**	0.0118

TABLE VI

IMPACT OF THE OPTIMIZER ON THE STRESS.

Learning rate	0.005	0.001	0.0005	0.0001
Stress GMF	0.0192	0.0173	0.0144	**0.0118**
Stress MLP	0.0168	0.0151	0.0144	**0.0117**

TABLE VII

IMPACT OF THE LEARNING RATE ON THE STRESS.

Impact of the sampling rate and NMF/NCF comparison We can see in Table VIII that the stress decreases with the sampling rate for all the architectures. Moreover, we can observe that the MLP is better than the GMF and that the overall stress remains smaller than 2%. However, we see that the NeNMF outperforms the NCF approaches when applied to the real-world dataset, which is in line with the findings of the comparative studies [11], [12]. Such result can be explained by the time stability of the real-data matrix combined to its high dimensions. These conditions can be in favor of a simple matrix product rather than a more complex model that needs to learn many additional parameters.

By comparing figure 7 and figure 8, one can notice that both reconstructions correspond to the original time series, but we can clearly see that at equal sampling rate, the GMF reconstruction is noisier than the MLP one.

Sampling rate	0.50	0.6	0.7	0.8	0.9
Stress GMF	0.0198	0.0186	0.0166	0.0166	**0.0158**
Stress MLP	0.0126	0.0170	0.0125	0.1210	**0.0119**
Stress NeNMF	0.0182	0.0117	0.0119	0.0116	**0.0115**

TABLE VIII

IMPACT OF THE NUMBER OF EPOCHS ON THE STRESS.

IV. CONCLUSION

NCF approaches enable us to achieve very low stress rate for the matrix completion task, both the simulated and real-world datasets. In this paper, we have studied extensively the impact of multiple parameters such as the optimizer, the learning rate or the number of epochs on the reconstruction quality. This allowed us to set an optimal training environment for the NCF models. The comparison with the NMF algorithms showed that NCF outperforms NeNMF on synthetic data, whereas this tendency is reversed when applied to Ripe Atlas data. One of the possible reasons is that we didn't use enough iterations, and we did not use the piecewise regularization. Finally, the addition of a regularization term didn't improve the completion

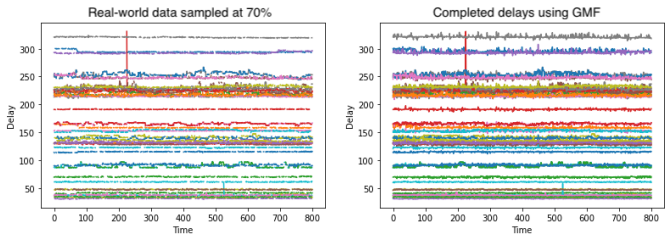

Fig. 7. Reconstruction of an 70% observed real-world delay matrix using GMF.

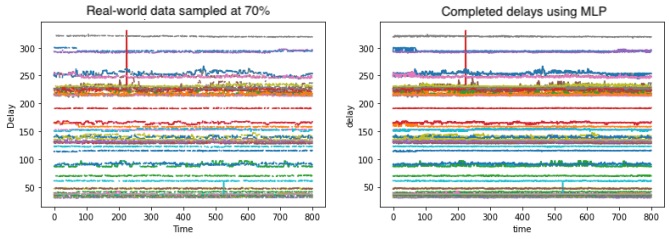

Fig. 8. Reconstruction of an 70% observed real-world delay matrix using MLP.

quality on the synthetic dataset. In the future, we consider exploring other regularization terms, as well as testing online versions of these methods for the matrix completion problem.

REFERENCES

[1] A. Ramlatchan, M. Yang, Q. Liu, M. Li, J. Wang, and Y. Li, "A survey of matrix completion methods for recommendation systems," *Big Data Mining and Analytics*, vol. 1, no. 4, pp. 308–323, 2018.

[2] E. J. Candès and B. Recht, "Exact matrix completion via convex optimization," *Foundations of Computational mathematics*, vol. 9, no. 6, pp. 717–772, 2009.

[3] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T.-S. Chua, "Neural collaborative filtering," in *Proceedings of the 26th international conference on world wide web*, 2017, pp. 173–182.

[4] Y. Koren, S. Rendle, and R. Bell, "Advances in collaborative filtering," *Recommender systems handbook*, pp. 91–142, 2022.

[5] S. Ghandi, A. Reiffers-Masson, S. Vaton, and T. Chonavel, "Non-negative matrix factorization for network delay matrix completion," in *7th IFIP/IEEE International Workshop on Analytics for Network and Service Management*, 2022.

[6] Synthetic data generation. [Online]. Available: https://github.com/Ghandisanaa/NMF-for-network-delay-matrix-completion/tree/main

[7] Ripe atlas. [Online]. Available: https://atlas.ripe.net/m

[8] S. Liu and Q. Wang, "Poster: online adaptive sampling for network delay measurement via matrix completion," in *2019 IFIP Networking Conference (IFIP Networking)*. IEEE, 2019, pp. 1–2.

[9] N. Seichepine, S. Essid, C. Fevotte, and O. Cappe, "Piecewise constant nonnegative matrix factorization," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2014.

[10] N. Guan, D. Tao, Z. Luo, and B. Yuan, "NeNMF: An optimal gradient method for nonnegative matrix factorization," *IEEE Transactions on Signal Processing*, vol. 60, no. 6, pp. 2882–2898, 2012.

[11] V. W. Anelli, A. Bellogín, T. Di Noia, and C. Pomo, "Reenvisioning the comparison between neural collaborative filtering and matrix factorization," in *Fifteenth ACM Conference on Recommender Systems*, 2021, pp. 521–529.

[12] S. Rendle, W. Krichene, L. Zhang, and J. R. Anderson, "Neural collaborative filtering vs. matrix factorization revisited," *CoRR*, vol. abs/2005.09683, 2020. [Online]. Available: https://arxiv.org/abs/2005.09683