Geometric Matrix Completion with Recurrent Multi-Graph Neural Networks

Antonin Barbe
Marc Boëlle
Zoé Herson
antonin.barbe@polytechnique.edu
marc.boelle@polytechnique.edu
zoe.herson@ens.psl.eu

Abstract

This report examines the matrix completion method proposed by Federico Monti, Michael M. Bronstein, and Xavier Bresson, which leverages geometric deep learning to incorporate additional information from the rows and columns of the matrix, represented as a graph. Their architecture outperforms state-of-the-art methods. After delving into the theory of graph convolutional neural networks, we investigated the importance of the graph structure and the method's sensitivity to it through several experiments. Our results show that the improvement in accuracy using the authors' graph, compared to a random graph, is surprisingly small; and that the recurrent part of the model can be replaced by a simpler architecture.
The original code from 2017 is available **on GitHub** but we decided to recode it in modern PyTorch for greater clarity and to conduct our own experiments. The code is available **by clicking on this link**.

Keywords

Matrix Completion, Multi-Graph Neural Networks, RNN

1 Introduction

Recommender systems have become an integral part of our daily lives, analyzing user behavior to provide personalized recommendations for goods and services. These suggestions significantly enhance the user experience on platforms such as streaming services or e-commerce sites. It is therefore unsurprising that companies like Netflix actively invest in research to improve the underlying models and algorithms.

A matrix completion problem is a great model for recommendation systems. It consists of a sparse matrix representing users as columns and items as rows, and the ratings are stored in the coefficients of the matrix [Monti et al.(2017)]. To suggest new items to the users, the model need to learn the empty coefficients of the matrix. This is a challenging problem, as the user-item matrix can reach substantial sizes and are usually quite sparse, due to the large number of users on the platform and items available. In the famous Netflix problem [Monti et al.(2017)] [Koren et al.(2009)], the matrix is of size is 480k movies × 18k users, and only 0.011 % of the entries are known. Very large matrices often lead to an increased number of parameters to learn, which increases in return the computation cost.

In this report, we go through the work of Federico Monti, Michael M. Bronstein and Xavier Bresson on geometric matrix completion with recurrent multi-graph neural networks.

2 Background on matrix completion

Matrix completion problems have been widely studied and a large number of methods have been proposed in order to solve them [Chen and Wang(2022)]. These methods range from traditional optimization methods such as convex optimization [Candès and Recht(2012)], matrix factorization [Koren et al.(2009)] or rank minimization [Lu et al.(2014)], to modern machine learning methods [Xue et al.(2017)]. Most of these methods take advantage of the fact that the matrix is low-rank, which means that most of the information is redundant.

Rank minimization methods focus on minimizing the rank function such that :

$$\arg\min_M h(M) = \mu f(M) + rank(M) \tag{1}$$

with f a differentiable loss function and rank is the rank function [Chen and Wang(2022)]. For matrix completion problems, f is defined as : $f(M) = \|M - M^*\|^2$, with M^* the reconstructed matrix. But this problem is NP-hard and computationally intractable. The rank function thus needs to be relaxed to surrogate functions such as nuclear norm [Monti et al.(2017)].

$$\arg\min_M h(M) = \mu f(M) + ||M||_* \tag{2}$$

Matrix factorization methods have a different approach of this problem. Their goal is to decompose the user-item matrix M into latent factors for both users and items in order to correctly capture their specificity. A famous algorithm following this method is the nonnegative matrix factorization (NMF) algorithm [Lee and Seung(1999)]. It decomposes the matrix M into $M = PQ$, where P profiles the user latent features and Q the item latent features. This methods is well adapted to very large matrix such as the one from the Netflix challenge. With (m, n) being the size of the matrix M and (m, r), (r, n) being the sizes of matrices P and Q respectively, this decomposition reduces the complexity from $O(mn)$ to $O(m + n)$.

The matrix factorization approach is guided by the **Singular Value Decomposition** (SVD). The distribution of singular values is key: the large singular values capture the most significant aspects of the matrix M. A matrix M can be approximated by retaining only

Geometric Data Analysis, December 2024, ENS Paris Saclay, M2 MVA

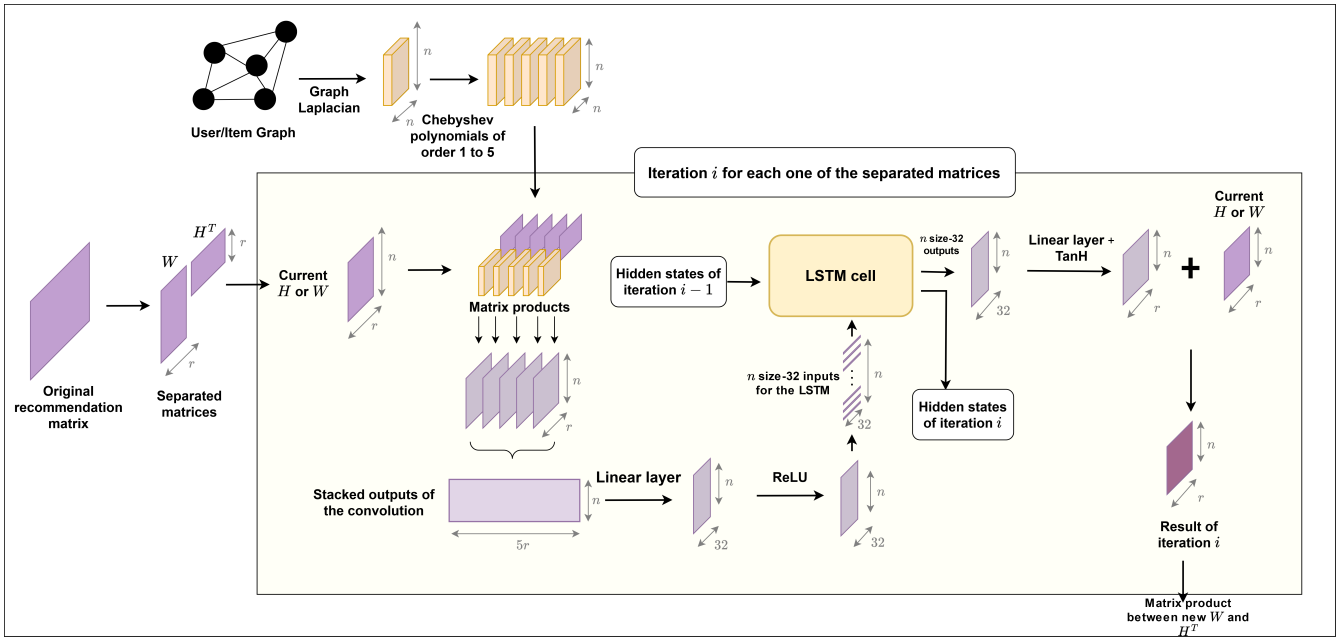


Figure 1: Layout of an iteration of the separable RGCNN model. On the left, matrix factorization is initialized with Singular Value Decomposition (SVD) algorithm : $X = U\Sigma V^T$. By only keeping the r highest singular values of Σ in Σ_r, with U_r and V_r the matrices with columns corresponding to the selected singular values, we choose $W = U_r\Sigma_r^{\frac{1}{2}}$ and $H = V_r\Sigma_r^{\frac{1}{2}}$. For the first iteration, the hidden states of the LSTM cell are fixed to 0.

the k largest singular values, enabling the construction of a rank-k approximation.

$$M = U\Sigma V^T \tag{3}$$

where $U \in \mathbb{O}_n$, $V \in \mathbb{O}_m$, and $\Sigma \in \mathbb{D}_{n,m}$.
With the emergence of machine learning, these methods were adapted to traditional learning algorithms but also modern deep learning [Chen and Wang(2022)] [Xue et al.(2017)]. The latent factors of matrix factorization can thus be turned into learnable parameters, but machine learning can also be applied to neighbors-based models (NBM) that compute similarities between user to recommend items according to the rating of other users.

Recently, to improve the performances of recommendation algorithms, there has been several attempts of adding relevant information such as similarity between items or users to the matrix completion models [Ma et al.(2011)] [Kuang et al.(2016)] [Rao et al.(2015a)]. In particular, graphs are well-suited to store that additional information. The work from Monti & al [Monti et al.(2017)] thus lays the foundations for enabling geometric deep learning on graphs to become a powerful tool regarding matrix completion problems.

3 Main content

3.1 Graph Spectral Analysis

A weighted graph G is a triplet (V, E, w), where V is the set of vertices of the graph, E is the set of edges, and w is the weight function that assigns a weight to each edge. A graph is usually represented by an adjacency matrix W, where W_{ij} is the weight of the edge between vertices i and j. This object is commonly used to model relationships between entities.

The matrix representation of a graph allows extracting information about the graph through its spectral decomposition. In particular, the object of interest is the **Laplacian** L, defined as $L = D - W$, where D is the diagonal degree matrix of each vertex. This matrix is positive semi-definite and corresponds to the transition matrix of a random walk on the graph.

Specifically, for a signal X indexed on the graph, the quadratic form associated with the Laplacian is:

$$X^\top LX = \sum_{(i,j)\in E} w_{ij}(x_i - x_j)^2,$$

where w_{ij} represents the weight between vertices i and j.

This expression measures how smooth the signal X is over the graph, with lower values indicating that the signal varies less between neighboring vertices.

For a matrix X containing r signals indexed on the graph, the value

$$\|X\|_G = \mathrm{Tr}(X^\top LX)$$

represents the sum of the quadratic forms for each of the r signals. This can also be expressed as:

$$\|X\|_G = \sum_{k=1}^{r} X_k^\top L X_k,$$

where X_k represents the k-th signal. This metric evaluates the smoothness of all r signals over the graph.
The norm induced by G on matrices, as defined below, ensures that the columns of X adhere to the neighborhood relationships defined by the graph.

Using this norm in our problem, the matrix completion task reduces to minimizing the following objective:

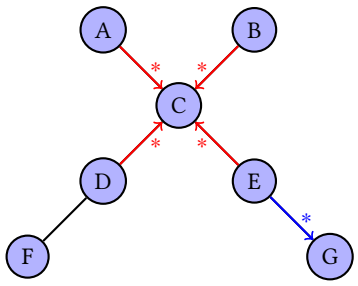
$$\|X\|_{G_r} + \|X^T\|_{G_c} + \frac{\mu}{2}\|\Omega \odot (X - Y)\|_F^2,$$

3.2 Graph Spectral Convolution Methods

3.2.1 Convolution Operation on a Graph. Convolution is a mathematical operation that merges two functions to produce a third, smoother function. In 1D, the convolution between a function f (input signal) and a function g (kernel or filter) is defined as:

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau) g(t - \tau)\, d\tau$$

Unlike mathematical convolution, where the kernel g is predefined, in CNNs, the filters are learned during training. This allows the network to adapt to specific data and optimize its performance for the given task.



Convolutions centered at C (red lines) and at G (blue lines)

Figure 2: Illustration of convolutions on a graph

Let us revisit the Laplacian of our graph to establish a similar operation on the graph. The Laplacian is a symmetric matrix that admits a **spectral decomposition**:

$$L = \Phi \Lambda \Phi^\top,$$

where Φ is the matrix of eigenvectors and Λ is the diagonal matrix of eigenvalues. This decomposition reveals the directions of variation in the graph, with each eigenvalue (frequency) corresponding to the elasticity of its associated direction of variation.

For a signal x defined on the graph, its spectral decomposition is given by

$$\hat{x} = \Phi^\top x,$$

where $\hat{x}$ represents the projection of x onto the eigenbasis of the graph Laplacian.

The **spectral convolution** of two functions x and y can be defined as the element-wise product of their respective Fourier transforms:

$$\widehat{(x * y)} = \widehat{x} \odot \widehat{y},$$

$$x * y = \Phi(\Phi^T x) \odot (\Phi^T y) = \Phi \mathrm{Diag}(\hat{y_1}, ..., \hat{y_n})\hat{x}$$

In general, to construct the convolution kernel, we decide to restrict the kernel's support to its neighborhood. For instance, a Gaussian kernel is primarily centered within the range -3σ to 3σ.
We aim to find filters that approximately follow this type of formula:

$$h(x) = \sum_{y \in \mathcal{N}(x)} g(x, y) \cdot f(y)$$

3.2.2 Intuition Behind the Basic CNN Layer. Unlike images, the notion of neighborhood relationships is more difficult to capture in graphs: the number of neighbors varies, and the intensity of the relationships between neighbors can differ significantly.
The Fourier transform allows us to identify the different scales of the graph. The eigenvectors associated with large eigenvalues correspond to the **local properties** of the graph. In contrast, the eigenvectors associated with small eigenvalues capture the **global properties** of the graph. (See the plot in the appendix.)
By transforming a signal x into a signal $\hat{x}$, we account for both the **local** and **global** characteristics of the signal on the graph, which are not immediately apparent from the raw signal.
By revisiting the formula for convolution, we observe that learning a filter amounts to finding the diagonal matrices $\hat{Y}_{lk}$, where l represents the output index and k the input index. The number of parameters for a layer in a CNN with q inputs and q' outputs is nqq'.

$$\tilde{x}_l = \xi\left(\sum_k \Phi \hat{Y}_{lk} \hat{x}_k\right).$$

However, this expression is computationally expensive $O(n^3)$ and results in excessive propagation across the graph: the information is no longer preserved locally.

3.3 Efficient spectral graph convolution networks

Let us improve the layers of our neural networks by addressing the two critical remarks mentioned earlier (computationally expensive and no longer local). To do so, we will restrict our filters to the space $\mathbb{P}_n[\tilde{L}]$.

3.3.1 Space of Polynomials. The adjacency matrix of a sparse graph remains sparse for small powers. This is because when raising the matrix to a power, the non-zero entries correspond to paths in the graph, and for graphs with few edges, these paths remain limited, preserving the sparsity of the matrix. This property, which

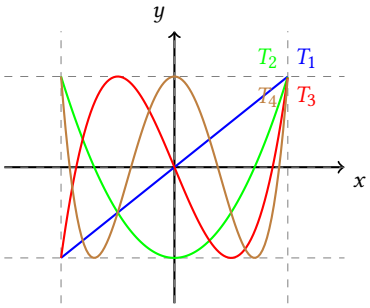


Figure 3: Graphical representation of the first four Chebyshev polynomials

is inherited by the Laplacian, prevents information from spreading across the entire graph and reduces the number of computations.

3.3.2 Filters Using Chebyshev Polynomials. The **Chebyshev Approximation Theorem** states that continuous functions can be approximated efficiently using Chebyshev polynomials.
Let $f : [-1, 1] \to \mathbb{R}$, there exists coefficients $c_0, \dots, c_{n-1}$ such that the polynomial $P(X) = \sum_{i=0}^{n-1} c_i T_i(X)$ is a polynomial that approximates f correctly on the interval $[-1, 1]$.
The coefficients are given by:

$$c_0 = \frac{1}{n} \sum_{k=0}^{n-1} f(w_k),$$

$$c_i = \frac{1}{n} \sum_{k=0}^{n-1} f(w_k) T_i(w_k)$$

where w_k are the roots of the Chebyshev polynomial T_{n-1}.

The Chebyshev abscissas are the best interpolation points for minimizing the oscillations of the interpolation polynomial, thus achieving the best possible convergence. Using this theorem, we can approximate various functions in $O(n^2)$ time while preserving local information.

Building on these results, we can represent spectral filters as an explicit expansion in the basis of Chebyshev polynomials.

$$\tau_\theta(\tilde{L}) = \sum_{j=0}^{n} \theta_j T_j(\tilde{L}) = \sum_{j=0}^{n} \theta_j \Phi T_j(\tilde{\Lambda}) \Phi^\top$$

where

$$\tilde{L} = 2\lambda_{\max}^{-1} L - I$$

This approach eliminates the need to compute the eigenvectors of the Laplacian. Since Chebyshev polynomials are computed recursively, this further reduces the computational cost of the method without significantly increasing the approximation error. (Chebyshev polynomials approximate functions on a segment while minimizing the Runge phenomenon.)
The Chebyshev polynomials $T_n(x)$ can be defined recursively as follows:

$$T_0(x) = 1, T_1(x) = x,$$

and for $n \geq 2$:

$$T_n(x) = 2xT_{n-1}(x) - T_{n-2}(x).$$

This section allowed us to construct the building block of a neural network that preserves local information (CNN).

3.4 Graph Convolutional Neural Networks

Recall that we aim to minimize the following norm:

$$\|X\|_{G_r} + \|X^T\|_{G_c} + \frac{\mu}{2} \|\Omega \odot (X - Y)\|_F^2,$$

where we want the rows and columns of X to respect the user graph and the movie graph, respectively. The **two-dimensional Fourier transform** is given by:

$$\hat{X} = \Phi_r^T X \Phi_c,$$

where Φ_r and Φ_c are the orthogonal eigenvector matrices of the row and column Laplacians.

This formula can be understood as follows: in the previous case, we only had a one-dimensional signal. Now, we aim to transform the rows in the spectral space of users and the columns in the spectral space of movies. For a given row, we recover the same formula as before.
The **spectral convolution** of two graphs can be rewritten as:

$$X * Y = \Phi_r (\widehat{X} \odot \widehat{Y}) \Phi_c^\top$$

A priori, a convolutional filter in this case requires $O(mn)$ parameters (the matrix Y).

3.4.1 Multi-graph convolution. Using the Chebyshev basis representation of row and column filters, the number of parameters can be reduced to $(p + 1)^2$:

$$\widehat{X} = \sum_{j,j'=0}^{p} \theta_{j,j'} T_j(\tilde{L_r}) X T_{j'}(\tilde{L_c}),$$

3.4.2 Separable convolution. As mentioned in the first section, one of the methods used to solve the matrix completion problem is to solve the following optimisation problem:

$$\min_{W,H,Z} \|WH^\top - Z\|_F^2 \quad \text{s.t.} \quad Z_\Omega = X_\Omega,$$

where $W \in \mathbb{R}^{m \times r}$, $H \in \mathbb{R}^{n \times r}$, and $Z \in \mathbb{R}^{m \times n}$.
The advantage of this method lies in the flexibility it provides to the approximation matrix by reducing its dimensionality. There is no longer a need to reduce the rank, as the flexibility introduced by the factors W and H automatically reduces the rank.

In our case, this reformulation of the problem allows us to separate it into two subproblems: one for the users and the other for the movies.

$$\tilde{w_{l'}} = \sum_{l'=0}^{q'} \sum_{j=0}^{p} \theta_{l,l',j}^r T_j(\tilde{L_r}) w_l$$

$$\tilde{h_{l'}} = \sum_{l'=0}^{q'} \sum_{j=0}^{p} \theta^{c}_{l,l',j} T_j(\tilde{L}_c) h_l$$

3.5 Matrix diffusion

The use of multiple CNN layers applied to the matrix X produces q outputs, each with the same size as the matrix. The next step is to merge these outputs to update our estimate of the matrix X.
The construction of the Graph Convolutional Network allows for the creation of local features on the matrix. This step can be seen as a complex embedding of the matrix with respect to the graph. The goal is now to leverage these features to iteratively adjust our matrix towards one of lower rank.
To achieve this, we proceed in iterations. At each iteration, we locally modify the factors W and H to integrate more information without increasing the rank.

$$H^{t+1} = H^t + dH^t$$
$$W^{t+1} = W^t + dW^t$$

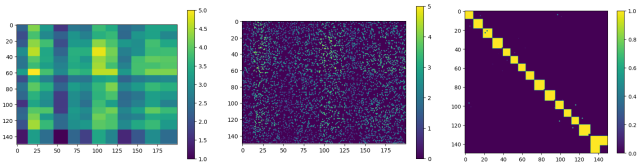
The original paper proposes using a recurrent neural network (RNN) to aggregate the information. We may wonder why the diffusion phenomenon is relevant at this stage of the problem. The use of an LSTM in our approach is justified by its ability to capture the evolving relationships between matrix elements during the diffusion process. The matrix completion task requires predicting small incremental changes at each step, without increasing the rank of the matrix. The iterative nature of the updates, where each step depends on the previous ones, mirrors the logic of solving puzzles like Sudoku, where each decision builds upon prior observations. The LSTM can be seen as a memory of past gradient directions, allowing it to build a better approximation by considering previous update attempts.

4 Results

4.1 Datasets presentation

The authors evaluate their approach on both real and synthetic data, mainly music and movie recommendation platforms. They show that with their new architecture, they outperform state-of-the-art techniques.
Synthetic dataset : The authors use a synthetic Netflix dataset containing 150 users and 200 movies, with a strong community structure : the users are clustered into 16 similar groups, and the movies in 12 groups (see 4c). The grades range from 1 to 5 stars, as shown in 4a. During training, we use 50% of the coefficients (see the training matrix in 4b) and the other 50% for evaluation.
Real datasets We decided to focus on one of the five real datasets the authors study, MovieLens 100K [Harper et al.(2016)]. This dataset consists in 1-5 stars rates for 1, 682 users and 943 users, with a density of 6, 3% known grades. For training, the authors use 80% of the known coefficients, and leave the remaining 20% for evaluation. We follow the same methodology for our experiments.

Graphs creation : In the model architecture, the user/item-graphs are of utmost importance, because it conditions the loss function which is minimized. However, as opposed to the synthetic dataset, there is no exact proximity graph, and assumptions have to be made to create one. The authors use the 10-nearest neighbor graphs in the space of user and movie features, following the work of [Rao et al.(2015b)]. More precisely, they map **features external to the matrix** in a higher-dimensional latent space, and construct a neighbor graph in this space. For the user-graph, the features are age, gender and occupation, mapped into a 22-dimension feature space for each user. For the movie graph, the features are binary values indicating the belonging to a specific genre, for a total of 18 genres. This choice of graph is crucial, as it implies the following hypothesis: **users with the same characteristics (age, occupation) tend to enjoy the same films, and films of the same genre tend to receive the same reviews.** As we shall see, the relevance of this graph is questionable.



(a) Original user-movie matrix **(b) Matrix used for model training** **(c) User adjacency matrix**

Figure 4: Characteristics of the synthetic Netflix dataset

4.2 Graph importance

Results reproduction : In all experiments, unless indicated otherwise, we use the same hyperparameters as in the paper : graph convolutions of order 5 Chebyshev polynomials 5 outputting 32 features, LSTM cells with 32 features and 10 diffusion steps, a 10^{-3} learning rate with Adam optimizer. For the separable method, we used a rank $r = 15$ for the synthetic dataset and $r = 10$ for MovieLens. For MovieLens, we add the result of RGCNN model. As shown in Figure 1 and 2, we obtain better results for both methods because the authors use the test RMSE associated with the lowest train RMSE, while we use the lowest test RMSE obtained. We note that the results between sRGCNN and RGCNN are closer than in their results. In view of the matrix M of rank 12 in Figure 4a, this seems coherent to us, since in this case the factorized form WH^T with H and W of rank at most $r = 15$ loses no information compared with the original matrix.

Graphs choice We decided to lead several experiments on both synthetic and MovieLens dataset in order to understand the contribution of graphs to RMSE. We led the following experiments:

(1) **Random graph** : creating random user/item-graphs with a given density d.
(2) **Graph perturbation** : probability p of adding or removing any edge starting from the authors' user/movie graphs.

For graphs containing randomness, we computed the mean RMSE out of 10 trials and indicated the corresponding standard deviation. The results are shown in Table 1 and 2.

MODEL	GRAPHS	RMSE	STD
GRALS (paper)	Original	0.0114	✗
RGCNN (paper)	Original	0.0053	✗
RGCNN (ours)	Original	0.0052	✗
sRGCNN (paper)	Original	0.0106	✗
sRGCNN (ours)	Original	0.0060	✗
sRGCNN	Random (d=0.1)	0.0253	3.6e-3

Table 1: Synthetic dataset - RMS error of different matrix completion methods. GRALS is the benchmark method [Rao et al.(2015b)] **that gives the authors the best baseline results.**

MODEL	GRAPHS	RMSE	STD
GRALS (paper)	Original	0.945	✗
sRGCNN(paper)	Original	0.929	✗
sRGCNN(ours)	Original	0.921	✗
RGCNN	Original	1.080	✗
sRGCNN	Random (d=0.1)	0.935	2.5e-3
sRGCNN	Perturbated ($p = 1/100$)	0.925	1.9e-3

Table 2: MovieLens dataset - RMS error of different matrix completion methods. GRALS is the benchmark method [Rao et al.(2015b)] **that gives the authors the best baseline results.**

Results : For the synthetic dataset, as expected, the choice of a random graph drastically increases the RMSE: from 0.0060 to 0.0253. This is because the original graph links together users and films whose notations are rigorously identical, so minimizing loss on these graphs leads to very good accuracy. With a random graph, we minimize the difference in ratings between users and films that don't necessarily belong to the same groups, hence the increase in RMSE.

For the MovieLens dataset, with a random graph and sRGCNN we obtain a slightly higher error than when using the author's user/item graphs (see Table 2). This shows that **the authors' method is already very successful without using information from graphs**, as the results with random graphs are still better than the best baseline method (GRALS). When perturbating the original graphs, we obtain intermediate results between those of original graphs and random graphs.

Such a good performance can be explained by matrix factorization: beyond simplifying their multi-graph convolution by two one-dimensional convolutions, it forces the resulting matrix to be of low rank $r = 10$. Moreover, the authors initialize their H and W matrices from the result of the SVD on the training recommendation matrix, a method traditionally used for matrix completion problems. The sRGCNN method can therefore be seen as an extension of this algorithm. The contribution of matrix factorization is reflected in the poor performance of RGCNN on MovieLens (results not shown in the original paper). With this method, train loss is lower than with sRGCNN, but the model fails to generalize to the test set (test RMSE of 1.08 vs 0.921 for sRGCNN). **In this case, the assumption of a low rank for the matrix gives better results while drastically speeding up training time.**

4.3 Ablation study: LSTM layer

In the original paper, the authors use a recurrent LSTM applied to feature vectors obtained from spectral convolutions on the matrix. They motivate their approach by recalling the efficiency of this architecture for learning data sequences properties, but do not go further in details. Plus, they do not compare their architecture with other possible methods.

In this context, we conducted experiments by replacing the LSTM layer by other architectures:

(1) a LSTM layer with hidden states of each iteration (see Figure 1) fixed to 0 (called "no memory" LSTM).
(2) a fully-connected linear layer (32 to 10)
(3) two successive fully-connected layers (32 to 32, 32 to 10) separated by a ReLU activation function.

The results we obtained are presented in Table 3.

ARCHITECTURE	RMSE
LSTM (Original)	0.921
LSTM (no memory)	0.926
Two linear layers	0.922
Linear layer	0.930

Table 3: RMS error of the separable model when replacing the LSTM layer by other architectures

Quite surprisingly, the "no memory" LSTM performance is really close to the original LSTM architecture. This means that the LSTM step does not take advantage of the fact that we're considering a sequence. Considering this result, we might wonder what the impact will be of completely replacing the LSTM with simple linear layers. Even more surprisingly, when we replace the LSTM with two fully connected linear layers separated by a ReLU activation function, we obtain a similar RMSE for a lower number of parameters (see Table 4). This shows that the LSTM architecture is not the most suitable for this problem.

ARCHITECTURE	PARAMETERS	VALUE
LSTM(Original)	$4(nm + n^2 + n) + md + d$	8650
Linear layer	$nd + d$	330
2 linear layers	$nm + m + md + d$	1386

Table 4: Comparison of number of parameters in the different architectures. We do not count the parameters of graph convolutions. For LSTM and the two successive layers, $n = 32$ is the input size, $m = 32$ the hidden state size, $d = 10$ the output size. For the linear layer, $n = 32$ is the input size, $d = 10$ the output size.

5 Conclusion

In this report, we went through the method proposed by Federico Monti, Michael M. Bronstein and Xavier Bresson in order to tackle

matrix completion problems using deep learning on graphs. We recalled that matrix completion is often used to formulate recommendation problems, and we delved into the theory of graph convolutional neural networks, on which the architecture of their method is based.

We then performed an extended study of the impact of the graph on the results, by testing other graphs such as random graphs with different densities and graphs obtained by perturbing the original graph. On a real-life movie dataset, we showed that with a random graph, the method performs already better than some state of the art methods, and that the additional information carried by the original graph only slightly improves the performances. When using a synthetic dataset whose graph provides reliable insight on node proximity, then the additional information becomes very relevant and the performance increases greatly. But this is not the case for most of the real dataset we want to extend the method to.

We also performed an ablation study where we changed the RNN architecture to observe the impact on the performances. We observed that the performance of LSTM is matched by a simple architecture of two successive linear layers, for a much smaller number of parameters. For this reason, we would have appreciated additional insights from the authors regarding the necessity of using an LSTM.

This enabled us to identify several promising research directions that we hope will be further explored in future work.

References

[Candès and Recht(2012)] Emmanuel Candès and Benjamin Recht. 2012. Exact matrix completion via convex optimization. *Commun. ACM* 55, 6 (June 2012), 111–119. https://doi.org/10.1145/2184319.2184343

[Chen and Wang(2022)] Zhaoliang Chen and Shiping Wang. 2022. A review on matrix completion for recommender systems. *Knowledge and Information Systems* 64, 1 (2022), 1–34.

[Harper et al.(2016)] F. Maxwell Harper, Joseph A. Konstan, and Joseph A. 2016. The MovieLens Datasets: History and Context. *ACM Trans. Interact. Intell. Syst.* 5 (2016), 19:1–19:19. https://api.semanticscholar.org/CorpusID:16619709

[Koren et al.(2009)] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix Factorization Techniques for Recommender Systems. *Computer* 42, 8 (2009), 30–37. https://doi.org/10.1109/MC.2009.263

[Kuang et al.(2016)] Da Kuang, Zuoqiang Shi, Stanley Osher, and Andrea Bertozzi. 2016. A Harmonic Extension Approach for Collaborative Ranking. arXiv:1602.05127 [cs.LG] https://arxiv.org/abs/1602.05127

[Lee and Seung(1999)] Daniel D Lee and H Sebastian Seung. 1999. Learning the parts of objects by non-negative matrix factorization. *nature* 401, 6755 (1999), 788–791.

[Lu et al.(2014)] Canyi Lu, Jinhui Tang, Shuicheng Yan, and Zhouchen Lin. 2014. Generalized nonconvex nonsmooth low-rank minimization. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 4130–4137.

[Ma et al.(2011)] Hao Ma, Dengyong Zhou, Chao Liu, Michael R. Lyu, and Irwin King. 2011. Recommender systems with social regularization *(WSDM '11)*. Association for Computing Machinery, New York, NY, USA, 287–296. https://doi.org/10.1145/1935826.1935877

[Monti et al.(2017)] Federico Monti, Michael M. Bronstein, and Xavier Bresson. 2017. Geometric Matrix Completion with Recurrent Multi-Graph Neural Networks. arXiv:1704.06803 [cs.LG] https://arxiv.org/abs/1704.06803

[Rao et al.(2015a)] Nikhil Rao, Hsiang-Fu Yu, Pradeep K Ravikumar, and Inderjit S Dhillon. 2015a. Collaborative Filtering with Graph Information: Consistency and Scalable Methods. In *Advances in Neural Information Processing Systems*, C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett (Eds.), Vol. 28. Curran Associates, Inc.

[Rao et al.(2015b)] Nikhil Rao, Hsiang-Fu Yu, Pradeep K Ravikumar, and Inderjit S Dhillon. 2015b. Collaborative Filtering with Graph Information: Consistency and Scalable Methods. In *Advances in Neural Information Processing Systems*, C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett (Eds.), Vol. 28. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2015/file/f4573fc71c731d5c362f0d7860945b88-Paper.pdf

[Xue et al.(2017)] Hong-Jian Xue, Xinyu Dai, Jianbing Zhang, Shujian Huang, and Jiajun Chen. 2017. Deep matrix factorization models for recommender systems.. In *IJCAI*, Vol. 17. Melbourne, Australia, 3203–3209.

Appendix

An LSTM cell consists of the following key equations:

- **Forget gate:**

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

  where f_t is the forget gate, W_f is the weight matrix for the forget gate, x_t is the input at time step t, h_{t-1} is the previous hidden state, and b_f is the bias term.
- **Input gate:**

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

  where i_t is the input gate, W_i is the weight matrix for the input gate, and b_i is the bias term.
- **Candidate memory cell:**

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

  where $\tilde{C}_t$ is the candidate memory cell, W_C is the weight matrix for the candidate memory, and b_C is the bias term.
- **Memory cell update:**

$$C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t$$

  where C_t is the memory cell at time t, and C_{t-1} is the previous memory cell state.
- **Output gate:**

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$

  where o_t is the output gate, W_o is the weight matrix for the output gate, and b_o is the bias term.
- **Hidden state update:**

$$h_t = o_t \cdot \tanh(C_t)$$

  where h_t is the hidden state at time t, and C_t is the updated memory cell.