Beyond : Geometric Matrix Completion with Recurrent Multi-Graph Neural Networks

Valentin Exbrayat, Hugo Pavy, Tom Mariani

ACM Reference Format:
Valentin Exbrayat, Hugo Pavy, Tom Mariani. 2025. Beyond : Geometric Matrix Completion with Recurrent Multi-Graph Neural Networks. In . ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Abstract

Recommender systems are central to modern digital platforms, yet their underlying user–item rating matrices are complex objects due to highly sparse and structured data. While classical collaborative filtering and low-rank matrix completion methods struggle to capture the relational geometry of users and items, recent graph-regularized approaches exploit smoothness on similarity graphs but remain shallow and scale poorly. The emergence of geometric deep learning has introduced expressive tools such as spectral and Chebyshev graph convolutions, enabling localized, size-independent filtering on graph-structured data. Building on these developments, Geometric Matrix Completion with Recurrent Multi-Graph Neural Networks (Monti et al., 2017 [5]) proposed a novel architecture that combines graph CNNs to extract locally stationary features and recurrent networks to model nonlinear rating diffusion. Although influential, this framework offers limited interpretability of its learned diffusion dynamics and relies on static and fully accurate user/item graphs. These limitations motivate a re-examination of geometric approaches to matrix completion and form the basis of the present work.

2 Context

Recommender systems have become indispensable to modern digital platforms, providing personalized suggestions in environments defined by overwhelming choice and information overload. From e-commerce and multimedia platforms to news feeds and social networks, these systems infer user preferences from massive, sparse, and often noisy interaction data. The key challenge lies in predicting missing user–item interactions despite sparsity, heterogeneity, and evolving user behavior. Classical paradigms such as collaborative filtering and content-based approaches (Breese et al., 2013 [1]; Pazzani and Billsus, 2007) remain foundational but do not fully leverage the rich relational structure that exists among users and items—structure that is crucial for improving generalization in sparse settings. These limitations have motivated the emergence of methods that explicitly integrate geometric and structural priors into the recommendation process, a direction that culminated in the Geometric Matrix Completion with Recurrent Multi-Graph Neural Networks framework introduced by Monti et al. (2017) [5].

Before geometric methods, the dominant line of research was low-rank matrix completion (Candès and Recht, 2008 [3]), which models user preferences as points within a shared low-dimensional latent space. Although effective, this approach assumes that interactions depend purely on latent factors and does not incorporate known similarities between users or between items. To address this gap, graph-regularized matrix completion introduced user and item similarity graphs and enforced smoothness priors, encouraging neighboring nodes in each graph to exhibit similar rating patterns (Rao et al., 2015). These methods brought geometric information into the problem but remained tied to linear models whose capacity is limited and whose parameterization scales with the number of users and items.

In parallel, the field of geometric deep learning opened the door to more expressive, nonlinear approaches capable of operating directly on irregular domains. Foundational developments in spectral graph theory demonstrated that convolution on graphs can be defined through the Fourier basis induced by the graph Laplacian (Bruna et al., 2014 [2]). Building on this, Chebyshev polynomial filters (Defferrard et al., 2017 [4]) provided computationally efficient, localized, and scalable graph convolutions with parameter counts independent of graph size—an essential property for recommender systems with millions of users and items. These advances enabled models to process graph-structured data in a manner analogous to CNNs on regular grids, while remaining sensitive to the underlying geometry of user and item relationships.

It is within this convergence of graph-based modeling and deep learning that Monti et al. (2017) [5] introduced Geometric Matrix Completion with Recurrent Multi-Graph Neural Networks. Their central contribution lies in treating the user–item rating matrix as data living on two interacting graphs (one for users and one for items) and learning representations through localized graph convolutional filters. This geometric viewpoint allows the model to extract fine-grained, spatially smooth features reflecting how preferences vary along each graph. Coupled with a recurrent module designed to model rating propagation as a nonlinear diffusion process, the approach blends spatial geometric structure with temporal dynamics. The resulting architecture represents the earliest, according to the authors, attempt to unify graph convolutional networks with recurrent neural models for matrix completion, offering a principled, parameter-efficient framework tailored to the relational nature of recommendation tasks.

3 Critical Explanation of the Paper

3.1 Theoretical Formulation

In this section, we summarize the theoretical framework introduced in [5] for geometric matrix completion with multi-graph neural networks. The goal is to reconstruct a partially observed matrix

Conference'17, Washington, DC, USA
© 2025 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-x-xxxx-xxxx-x/YYYY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

$Y \in \mathbb{R}^{m\times n}$ from entries indexed by $\Omega \subseteq [m] \times [n]$, while leveraging structural information encoded in graphs over rows and columns.

3.1.1 Rank Minimization and Graph-Regularized Formulation

The original problem is formulated as a rank minimization:

$$\min_X \ \mathrm{rank}(X) \quad \text{s.t. } x_{ij} = y_{ij}, \quad \forall (i,j) \in \Omega. \tag{1}$$

Since rank minimization is NP-hard, the authors [?] propose a convex surrogate incorporating smoothness constraints with respect to row and column graphs:

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

where $\|X\|^2_{G_r} = \mathrm{tr}(X^\top \Delta_r X)$ and $\|X\|^2_{G_c} = \mathrm{tr}(X\Delta_c X^\top)$, with Δ_r and Δ_c being the Laplacians of the row and column graphs,respectively. In this setup graphs are seen as adjacency matrix. This formulation encourages the reconstructed matrix to be smooth with respect to the underlying graphs.
An alternate resolution used to explicitly enforce low-rank structure tries to factories the matrix as $X = WH^\top$, leading to

$$\min_{W,H} \ \frac{1}{2}\|W\|^2_{G_r} + \frac{1}{2}\|H\|^2_{G_c} + \frac{\mu}{2}\|\Omega \circ (WH^\top - Y)\|^2_F, \tag{3}$$

where $W \in \mathbb{R}^{m\times r}$ and $H \in \mathbb{R}^{n\times r}$. These regularization terms still encourage the factors to vary smoothly along the respective graphs.

3.1.2 Spectral Graph Filtering

The authors describe filtering along the graphs using the spectral domain. As Laplacians are positive semi-definite matrices, we let the Laplacians be decomposed as

$$\Delta_r = \Phi_r \Lambda_r \Phi_r^\top, \qquad \Delta_c = \Phi_c \Lambda_c \Phi_c^\top.$$

The graph Fourier transform of X is $\widehat{X} = \Phi_r^\top X \Phi_c$. A spectral filter with multiplier $\widehat{Y}$ acts element-wise on $\widehat{X}$:

$$X \star Y = \Phi_r(\widehat{X} \circ \widehat{Y})\Phi_c^\top, \tag{4}$$

where $\circ$ denotes element-wise multiplication. This formulation is explicitly presented in the paper and highlights how the filtering operation combines information from row and column graph spectra.

3.1.3 Chebyshev Polynomial Expansion and Parameter Interpretation

The paper approximates the spectral multiplier $\widehat{Y}$ using a truncated bivariate Chebyshev polynomial expansion:

$$\tau(\widetilde{\lambda}_r, \widetilde{\lambda}_c) = \sum_{p,q=0}^{P} \theta_{pq}\, T_p(\widetilde{\lambda}_r)\, T_q(\widetilde{\lambda}_c), \tag{5}$$

where T_p denotes the Chebyshev polynomial of degree p, and $\Theta = (\theta_{pq})$ is the matrix of coefficients. Since Chebyshev polynomials form an approximation basis over $[-1, 1]$, this representation allows the model to approximate any smooth spectral filter and truncating the expansion reduces the number of parameters from $O(mn)$ (the full spectral multiplier) to $(P+1)^2$, making the approach computationally tractable.

This function expresses the spectral filter as a polynomial of the eigenvalues of the row and column Laplacians. Each coefficient θ_{pq} controls the amplitude assigned to a specific pair of graph frequencies: the p-th frequency of the row graph and the q-th frequency of the column graph. This interpretation follows from the graph Fourier transform.(see Equation 4). The eigenvectors form an orthogonal basis capturing oscillatory patterns on the graphs, while the eigenvalues $\widetilde{\lambda r}, i$ and $\widetilde{\lambda c}, j$ correspond to graph frequencies: small eigenvalues encode smooth, slowly varying structures, whereas large eigenvalues encode rapid, localized variations.

Convolution in the spectral domain is defined as multiplication, so its effect is entirely determined by how each spectral component is amplified or attenuated. Crucially, this amplification depends only on the frequencies, not on the signal's orientation in the vertex domain. Hence, the convolution operator seems fully described by the scalar function $\tau(\widetilde{\lambda r}, i, \widetilde{\lambda c}, j)$, evaluated at pairs of eigenvalues.

Using standard functional calculus, this spectral multiplication is equivalent to polynomial filtering in the vertex domain:

$$\widetilde{X} = \sum_{p,q=0}^{P} \theta_{pq}\, T_p(\widetilde{\Delta}_r)\, X\, T_q(\widetilde{\Delta}_c), \tag{6}$$

This shows that filtering can be performed efficiently without explicit eigenvector computations while preserving the dependency on the graph spectra via Chebyshev polynomials.

3.1.4 Recurrent Neural Network Component

The framework further introduces a recurrent neural network that operates on these filtered signals. At each step, the RNN updates the state by combining the current filtered matrix with previous states, enabling propagation of information across multiple graph supports and modeling nonlinear interactions. This LSTM-based implementation could likely be enhanced using more advanced architectures such as Transformers, but we chose to focus on the geometric aspects of the problem.

3.2 Implementations

3.2.1 The existing code

The paper comes with an existing implementation written in Python 2 and TensorBoard, designed to run in notebooks. Our first proposal is to reimplement the model with clean, modular code. You can find our code at this link: *github.com/hpavy/multi_graph_cnn*. We chose to use Poetry to manage the Python environment and ensure reproducibility. We structured the code into separate modules (training, model, main, data) to make it as easy to understand as possible. While the core code is in Python files, we only use notebooks for plotting graphs. Our implementation also supports GPU acceleration, making it faster.

The implementation of the paper is available in the src folder. The GitHub repository also includes separate folders for the tests and experiments conducted with the proposed model.

To validate our implementation, we can verify that our results match those reported in the paper. We implemented both models: the factorized version and the standard version.

Table 1: Comparison of RMSE results between the paper and our implementation.

Model	Dataset	Paper RMSE	Our RMSE
Normal	Synthetic Netflix	0.0053	0.0064
Factorized	MovieLens	0.929	0.939

For both models, the hyperparameters are available in the original Python 2 implementation. This ensures that we can reproduce the same results. To modify the hyperparameters in our code, you can edit the config.yaml file.

3.3 Normal model

For the normal model, we evaluated our implementation on the Synthetic Netflix dataset. The paper reports a test RMSE of 0.0053 (Table 2 of the original paper), while our implementation yields an RMSE of 0.0064 (as shown in Table 1). Despite this slight difference, the close match validates our implementation of the normal model. It is worth noting that these RMSE values, on the order of 10^{-3}, are exceptionally low for rating prediction tasks typically ranging between 1 and 5.

3.4 Factorized model

For the factorized model, we evaluated our implementation on the MovieLens dataset. The results reported in the paper (Table 4 of the original paper) show an RMSE of 0.929 for the test dataset, while our implementation achieves an RMSE of 0.939 (as shown in Table 1). This close match validates our implementation of the factorized model.

3.5 Visualization and Step back

In Fig.1, we observe the difference between the true values and the model's predictions. The comparison shows that the two matrices are extremely close. This is expected: for ratings ranging from 1 to 5, the RMSE is only 0.006 and the maximum error is around 0.1. Such precision would be impossible to achieve on a real dataset, where the measurement uncertainty is significantly higher.

4 Further Analysis

4.1 Analysis of the Synthetic Netflix Dataset

4.1.1 Dataset Structure and Graph Topology

To evaluate the performance of the Multi-Graph CNN (MGCNN), we need to inspect the structure of the "Synthetic Netflix" dataset. In fact, we know that this dataset was introduced as an easy case of matrix completion and therefore we need to understand what in means to have such a RMSE metric. This dataset comprises the completed matrix of film rankings per user, which are numbers between 1 and 5 and two distinct graphs: a user similarity graph ($\mathcal{G}_r$) and an item(in this case films) similarity graph ($\mathcal{G}_c$). To understand the links within users and films, we analyze the adjacency matrices and the spectral properties of their normalized Laplacians, see Figure 3, Figure 4, Figure 5 and Figure 6.

Despite a few minor irregularities, both graphs clearly exhibit multiple connected components: we observe 15 components in the

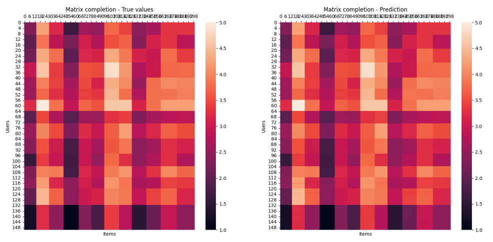


Figure 1: Comparison between True and Prediction values of matrix completion on synthetic Netflix dataset

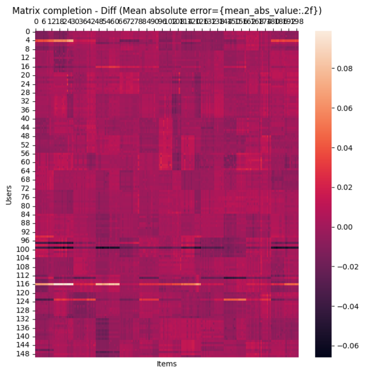


Figure 2: Difference between True and Prediction values of matrix completion on synthetic Netflix dataset

user graph and 12 in the film graph [Figure 1]. According to spectral graph theory, the number of connected components of a graph is equal to the multiplicity of the eigenvalue 0 in the spectrum of its (unnormalized or normalized) Laplacian matrix. This confirms the initial intuition obtained from the visual inspection of the adjacency matrices.

Furthermore, the presence of the eigenvalue 1 in the normalized Laplacian spectrum provides additional insight, as it often reflects redundancy or duplicated structures in the graph, such as repeated user profiles or duplicate film nodes.

For the rescaled Laplacian, they considered the transformation

$$\tilde{\Delta} = \frac{2}{\lambda_{\max}}\Delta - I_n,$$

where $\lambda_{\max}$ denotes the largest eigenvalue of the Laplacian. In their implementation, they used the simplified form

$$\tilde{\Delta} = \Delta - I_n,$$

originally justified by the fact that one may expects $\lambda_{\max} \approx 2$, making the full rescaling unnecessary. Therefore we tried to correct this simplification by re-running the training with the real rescaled Laplacian, however the obtained model reaches an RMSE of 0.0085, indicating relatively the same predictive accuracy. The pipeline was only run once with the hyperparameter given by the paper.

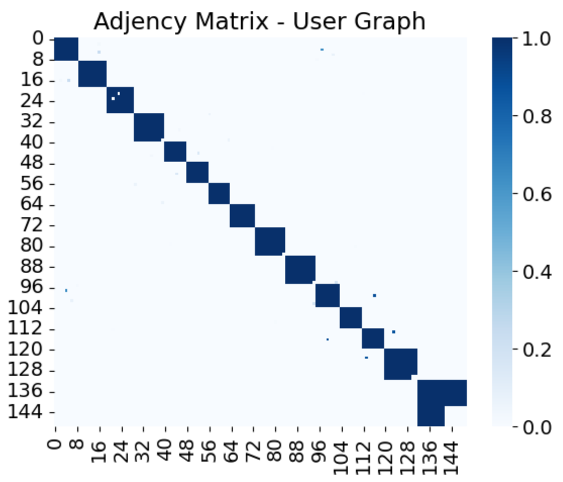


Figure 3: Heatmaps of Adjacency Matrices for User ($\mathcal{G}_c$). Block diagonal structures indicate strong clusters.

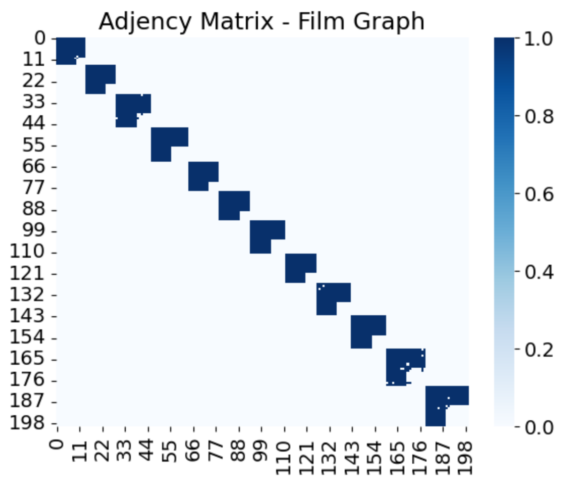


Figure 5: Heatmaps of Adjacency Matrices for Item ($\mathcal{G}_r$) graph.

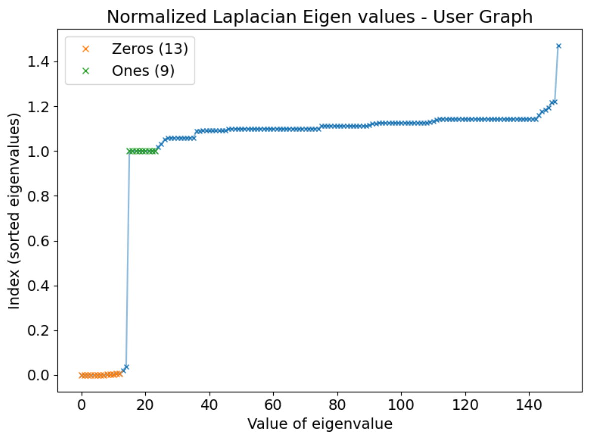


Figure 4: Spectrum of Normalized Laplacian Eigenvalues for users

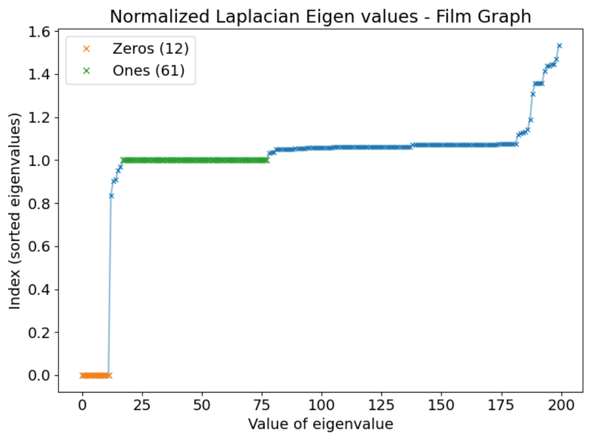


Figure 6: Spectrum of Normalized Laplacian Eigenvalues for films

4.2 Understanding the impact of the graph.

We have observed that the values used in the paper are very similar. Thus, with a perfect graph, it is of course easy to find the correct values for our matrix completion.
We would like to estimate to what extent the graph needs to be perfect. Indeed, one might expect that if a poor graph fails to yield good results, in practice the graph will never be perfect for the task we aim to solve.

4.2.1 The synthetic dataset.

In order to assess the impact of the graph, we need to quantify how perfect the graph is. To achieve this, we propose creating a synthetic graph from scratch. This allows us to systematically degrade the graph by reducing the information it contains.,

Our approach is to construct the simplest possible interpretable graph: For the problem of users ranking movies, we create two clusters of users and two clusters of movies. Each user prefers one type of movie or the other.

We define the user cluster of size n_{user} as: $C = [m_1, m_2, \ldots, m_{n_{user}}]$, where $m_i = 0$ if user i prefers the first type of movie (e.g., action movies) and $m_i = 1$ if the user prefers the second type.

From this cluster, we generate the user graph. We introduce a parameter, p_{within}, to measure the quality of the graph. This parameter represents the probability that a randomly chosen neighbor of a user belongs to the same cluster as that user.

- If $p_{within} = 1$ that means that our graph is perfect.
- The closer p_{within} is from 0.5, the poorest the graph is.
- If $p_{within} = 0.5$, we have the random graph.

- If $p_{within} < 0.5$, we have a graph that is giving wrong information.

Then from this cluster, we create the user graph.

4.2.2 Assessing the Influence of the Graph

We will use this synthetic dataset to evaluate how the graph influences the training process. To this end, we will train the model using the same cluster structure, varying only the p_{within} parameter. We will test values of p_{within} ranging from 1 down to 0.3.

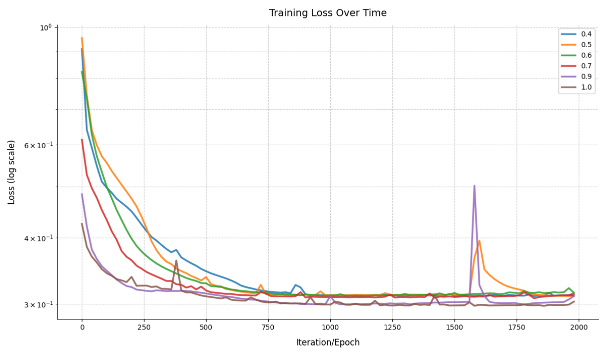


Figure 7: Training loss over epochs (log scale).

4.2.3 Interpretation

The results of these experiments are quite insightful. As expected, we observe that a better graph leads to faster model convergence and a lower final loss.

However, the differences are not substantial: all models converge to nearly the same loss value. Since we select ranks in the ranking matrix using a uniform distribution with a standard deviation of approximately 0.29, an error of around 0.3 indicates that the model performs well.

Surprisingly, even with a random graph ($p_{within} = 0.5$) or a graph containing mostly incorrect information ($p_{within} = 0.4$), the model still achieves good results. We hypothesize that, by minimizing the loss, the model learns an embedding for the users and effectively assesses the reliability of the graph. Thus, if the graph is unreliable, the model simply disregards the user graph. Given that the movie graph remains accurate and sufficient ranking data is available, the model can still learn effectively. This demonstrates the model's resilience to a poor user graph.

With the synthetic dataset implementation complete, a natural next step would be to investigate how adding a similarly poor movie graph (on top of the poor user graph) might further impact the training process.

4.2.4 Assessing the Influence of Two Poor Graphs

We will replicate the previous experiment, but this time we will vary the p_{within} values for both the user and movie graphs. We will test values of p_{within} ranging from 1 down to 0.3 for both graphs.

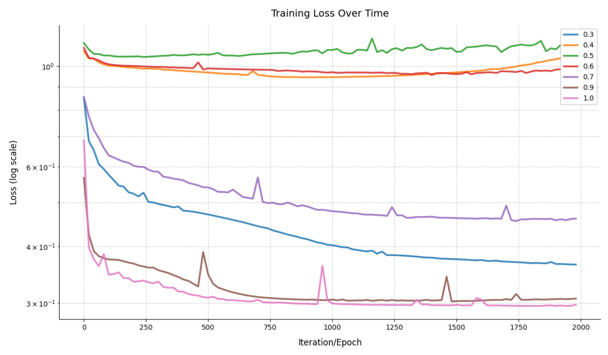


Figure 8: Training loss over epochs (log scale).

4.2.5 Interpretation of the Influence of Two Poor Graphs

As expected, the poorer the graphs are, the less the model converges effectively. There is a clear threshold between $p_{within} = 0.6$ and $p_{within} = 0.7$; below this threshold, the model loses significant performance.

These results confirm that the model is resilient to one poor graph. This is a particularly insightful finding for practical applications. For instance, in the context of user-movie ranking, constructing a reliable movie graph is relatively straightforward. Thus, even if learning a user graph is challenging, the model should still perform well.

4.3 Interpreting the Coefficients of the Chebyshev Filter

We begin with the central question: can we provide a clear spatial interpretation of the coefficients θ_{pq} in the expression

$$\widetilde{X} = \sum_{p,q=0}^{P} \theta_{pq}\, T_p(\widetilde{\Delta}_r)\, X\, T_q(\widetilde{\Delta}_c)$$

While this formulation is natural in the spectral domain, the Chebyshev terms $T(\widetilde{\Delta})$ are difficult to interpret directly on the graph.

However, powers of the (true) Laplacian matrix admit a clear structural meaning. Applying the Laplacian to a graph signal produces

$$(Lx)_i = \sum_{j \in \mathcal{N}(i)} w_{ij}(x_i - x_j),$$

which shows that Lx aggregates information from the 1-hop neighborhood of node i. More generally, $L^k x$ aggregates information up to the k-hop neighborhood. From matrix multiplication rules, $(L^k)_{ij}$ is nonzero if and only if a path of length k exists between nodes i and j. Thus, powers of L provide a natural and interpretable measure of multi-hop diffusion on the graph.

4.3.1 Transferring the Filter to the Polynomial Basis

To obtain interpretable diffusion weights, we expand the Chebyshev polynomials and rewrite the filter as a sum of Laplacian powers. Let d_{ab} denote the coefficients after expressing the Chebyshev filter in the monomial basis:

$$\widetilde{X} = \sum_{a,b} d_{ab}\, \widetilde{\Delta}_r^a\, X\, \widetilde{\Delta}_c^b.$$

The coefficients d_{ab} are computed by writing $T_p(X) = \sum_a \gamma_{p,a} X^a$:

$$d_{ab} = \sum_{j=a}^{p} \sum_{j'=b}^{p} \theta_{jj'} \gamma_{j,a} \gamma_{j',b} \tag{7}$$

These coefficients d_{ab} quantify the contribution of a-hop diffusion along rows and b-hop diffusion along columns, but they still rely on the rescaled Laplacian $\widetilde{\Delta}$, whose spectral meaning is less intuitive.

4.3.2 Recovering the True Laplacian Coefficients

To recover spatially meaningful diffusion coefficients, we convert from the rescaled Laplacian $\widetilde{\Delta}$ to the true Laplacian Δ. Using the affine relation

$$\widetilde{\Delta} = \alpha\Delta - I, \qquad \alpha = \frac{2}{\lambda_{\max}},$$

and applying the binomial theorem, we obtain

$$(\alpha\Delta - I)^a = \sum_{k=0}^{a} \binom{a}{k} (\alpha\Delta)^k (-1)^{a-k}.$$

Substituting this into the polynomial expansion yields final coefficients c_{kl} in the true Laplacian basis,

$$\widetilde{X} = \sum_{k,l} c_{kl}\, \Delta_r^k\, X\, \Delta_c^l.$$

These coefficients c_{kl} quantify the actual k-hop (rows) and l-hop (columns) diffusion weights and are given by :

$$c_{kl} = \alpha_r^k \alpha_c^l \sum_{a=k}^{p} \sum_{b=l}^{p} d_{ab} \left[\binom{a}{k} (-1)^{a-k} \right] \left[\binom{b}{l} (-1)^{b-l} \right]$$

After a training, we can look at a heatmap of these coefficients

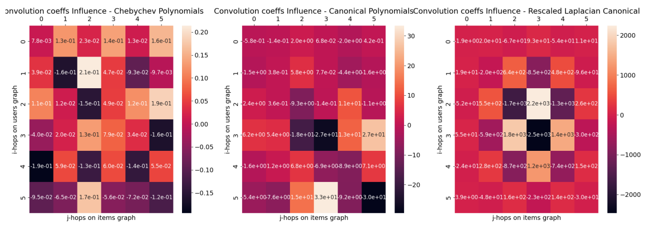


Figure 9: First convolutional filter coefficients

(a) left: original coefficients in front of Chebychev polynomials, middle: transferred coeffciients in front of canonical polynomials, right: transferred to real Laplacian

4.3.3 Stability Considerations

Because the transformation from Chebyshev coefficients θ_{pq} to the true spatial coefficients c_{kl} involves nested polynomial expansions and alternating signs, it may introduce numerical instability, that may explains why we have this peak around (3,3). We provide in Appendix 1 a visual explanation of what could have happened with this transformation. This consideration deeply challenges the given transformation, we should consider to apply regularization or

4.4 Influence of k-Hop Neighbours

In the previous section, we tried to understand the coefficients of the convolution alone and suggested a spatial understanding for a path up to k neighbors, meaning a walk of k steps. We now address a second question: how can we precisely quantify the influence of k-hop neighbors on the filtered output? To do so, we study how perturbations in the input entry x_{uv} affect the filtered output entry $\widetilde{x}_{ij}$.

Starting from the polynomial filter representation, we can write for a given (i, j) by developing the matrix multiplication

$$\widetilde{x}_{ij} = \sum_{r,c} \theta_{rc} \sum_{k,l} [T_r(\widetilde{\Delta}_r)]_{ik}\, x_{kl}\, [T_c(\widetilde{\Delta}_c)]_{jl},$$

we differentiate with respect to x_{uv} and obtain the influence measure

$$\text{Influence}(u, v \to i, j) = \frac{\partial \widetilde{x}_{ij}}{\partial x_{uv}} = \sum_{r,c} \theta_{rc}\, [T_r(\widetilde{\Delta}_r)]_{iu}\, [T_c(\widetilde{\Delta}_c)]_{jv}.$$

This expression captures the combined effect of row and column diffusion through the Chebyshev operators.

4.4.1 Restricting to Exact k-Hop Neighbourhoods

To interpret the influence spatially, we identify all pairs (u, v) such that node i is at exactly distance k from node u, and node j is at exactly distance l from node v. Although one could evaluate the influence for each individual pair (u, v), it is more informative to aggregate these values.

$$E_{k,l} = \frac{1}{N(k,l)} \sum_{i} \sum_{u \in \mathcal{V}^k(i)} \sum_{j} \sum_{v \in \mathcal{V}^l(j)} \left| \frac{\partial \tilde{x}_{ij}}{\partial x_{uv}} \right|$$

We define $\mathcal{V}^k(i)$ as all neighbors of i for which the shortest path length is k and $N(k, l)$ as the number of terms in this sum. This yields a 2D energy map indexed by (k, l), which describes the strength of diffusion as a function of neighbourhood depth. We decided to take the absolute value of the influence to avoid the cancellation problem but in our case it would be also interesting to see if all k-hop neighbors have a negative or positive influence on this filter.

4.4.2 Energy Visualization

The resulting matrix can be visualized as a heatmap where the row index corresponds to the distance from i and the column index to the distance from j. This plot provides a direct interpretation of how much the filter relies on information from neighbors at different hop distances along the two graph dimensions.

This figure leads us to wonder if we are able to reduce the max degree of the polynomial, chosen one was 5. We can introduce a mixed degree polynomial equation and differentiate the max degree of row polynomials p_r and column ones p_c. The equation becomes :

$$\widetilde{X} = \sum_{p=0}^{p_r} \sum_{q=0}^{p_c} \theta_{pq}\, T_p(\widetilde{\Delta}_r)\, X\, T_q(\widetilde{\Delta}_c)$$

We can therefore choose $p_r = 1$ and $p_c = 2$ as suggested by Figure 6. This result in a RMSE of 0.0098 on the synthetic Netflix dataset. Despite a small drop in performance (compared to 0.0064, see Table

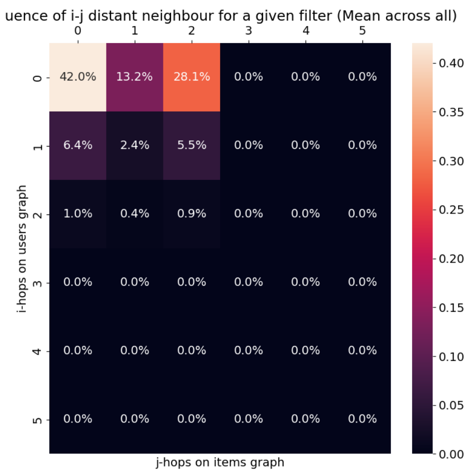


Figure 10: Mean Energy over all (32) filters for the same training as in the paper

1), we showed that our visualization brings a good suggestion to determine the degree of polynomials.

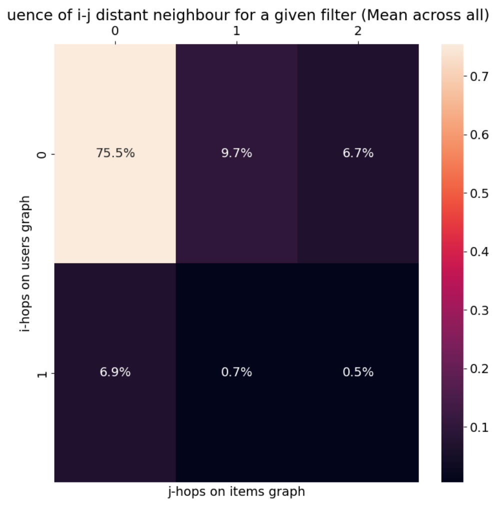


Figure 11: Mean Energy over all (32) filters for $p_r = 1$ and $p_c = 2$

4.5 Representation of the Learned User and Movie Embeddings

4.5.1 The Idea

From the previous sections, we observed that the model combines two graphs together with a ranking matrix in order to complete the overall ranking task. In the part focusing on the influence of the graphs, we showed that the model is capable of extracting relationships between users. If the graphs produce meaningful clusters, even if these clusters are inverted or only partially correct, the model is still able to leverage them. Otherwise, it learns to ignore the noisy graph and relies on other relational signals to complete the sparse input.

With this in mind, we can view the model as a tool for learning *new embeddings* for both users and movies, particularly when using matrix factorization methods that decompose

$$X = WH^\top.$$

These embeddings are well suited for capturing relationships between users or movies that are specifically tailored to the matrix-completion task.

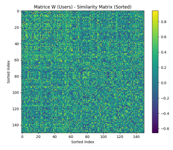
4.5.2 The Experiment

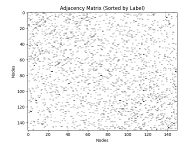
In this experiment, we investigate the information encoded in the factorized output matrices, depending on the quality of the user and movie graph clusterings. Our goal is to determine whether the learned user embedding W is capable of producing a good clustering of users, even when the original user graph is poorly constructed.

Using the synthetic data generation framework, we construct:

- a **poorly clustered user graph** with $p_{\text{within}} = 0.6$,
- a **well-clustered movie graph** with $p_{\text{within}} = 0.9$.

From earlier experiments, we know that the model is still able to complete the matrix accurately in such settings. To examine the learned user structure, we compute the cosine similarity between the rows of W, thereby producing an adjacency matrix where each entry (i, j) represents the similarity between users i and j. We find that, after training, the users are correctly clustered. The corresponding figure (not shown here) compares the adjacency matrices of the initial, untrained W and the final, learned W under this poor user-graph setting.


(a) Adjacency Matrix from the W before training


(b) Adjacency Matrix of the user graph used for learning

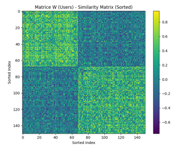

(c) Adjacency Matrix from the W after training

Figure 12: Comparison of adjacency matrices.

To evaluate how suitable an adjacency matrix is for clustering, we compute its normalized Laplacian and analyze the eigenvector associated with the smallest non-zero eigenvalue, which captures the principal global partition of the graph. We then compare the resulting clustering with the ground-truth user groups. The results are summarized in the following table:

We observe that the final embedding W achieves **perfect clustering of the users**, despite the poor quality of the input user graph. The combination of ranking information and a well-clustered movie graph is sufficient for the model to reconstruct high-quality user clusters.

This observation raises an important question: if the model is able to learn embeddings that outperform the fixed, pre-constructed

Metric	Accuracy
accuracy_u_init	0.5133
accuracy_m_init	1.0000
accuracy_W_init	0.6133
accuracy_H_init	0.5500
accuracy_W	1.0000
accuracy_H	1.0000

Table 2: Accuracy values for initial and learned embeddings.

user and movie graphs, should these graphs remain fixed during training? Since the learned embeddings effectively produce better graph structures over time, it may be beneficial to allow the graphs used in the convolutional computations to evolve dynamically, potentially improving matrix completion performance even further.

5 Conclusion

In this report, we approached matrix completion as a geometric deep learning problem on graphs and successfully reproduced and reimplemented the framework of Monti et al. (2017) [5]. We also analyzed the synthetic Netflix dataset revealing doubtful metrics, which motivated the creation of our own synthetic dataset, reducing the number of groups but offering full control over both the graph structure and the underlying rating matrix.

One of the outcome of this work is a clearer understanding of the convolutional layer used in geometric matrix completion. This insight is essential for designing stronger graph-based models. Although other approach can emerge, we developed a set of analytical tools that can support future architecture design. Another important outcome of our work is a first understanding of how the matrix and the graph jointly combine information to separate users into coherent groups.

The perspectives are to test these tools on real-world datasets, try to improve graph structures specially for matrix completion, potentially learned during training from user and item features, and to develop multi-scale reasoning mechanisms capable of capturing long-range dependencies in complex graphs.

References

[1] John S. Breese, David Heckerman, and Carl Kadie. Empirical analysis of predictive algorithms for collaborative filtering, 2013.
[2] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs, 2014.
[3] Emmanuel J. Candes and Benjamin Recht. Exact matrix completion via convex optimization, 2008.
[4] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering, 2017.
[5] Federico Monti, Michael M. Bronstein, and Xavier Bresson. Geometric matrix completion with recurrent multi-graph neural networks, 2017.