Evaluating the performance of sparsified precision VNNs as a graph collaborative filter

Jort Boon

Supervisors: Elvin Isufi[1], Andrea Cavallo[1] & Chengen Liu[1]

[1]EEMCS, Delft University of Technology, The Netherlands

Abstract

Graph Neural Networks (GNNs) are an effective architecture for implementing collaborative filtering-based recommender systems. This paper evaluates the performance and computational complexity of precision matrix-based VNNs as a collaborative filter on the MovieLens-100K dataset. Results show the estimated precision matrix contains a high amount of noise when calculated from sparse data, which impacts the performance of the model. After sparsifying the precision matrix, the performance and computational complexity improved significantly.

1 Introduction

Recommender systems [1] are an essential part of many modern applications, such as social media apps, media streaming services, e-commerce, and many more. One of the most successful approaches when building recommender systems is Collaborative Filtering (CF) [2]. CF utilizes known features from similar users to predict unknown features for a target user. Due to the high-dimensional and sparse nature of many user-preference datasets, model-based CF implementations are preferred, as they reduce the data dimensionality to meet memory and computational constraints. Traditional dimensionality reduction techniques such as Principal Component Analysis (PCA) [3] fail to capture non-linear relationships and are unstable when the eigenvalues of the components are close. Graph Neural Networks (GNNs) [4] mitigate these problems by using a neural network approach for modeling non-linear relationships on graph data. With the modeled GNN, a Collaborative Filter can be created by gathering information from neighboring nodes (users) and using this information when making new predictions. One approach to building such a GNN is by using the precision matrix to model relationships between users. The precision matrix defines the conditional interdependence between users, which is a crucial metric for determining user relationships. This paper will examine the performance and computational complexity of this precision matrix approach, and how sparsifying this matrix affects both metrics.

2 Background

2.1 Graph Neural Networks

Most social behavior can be represented as graphs naturally, where nodes (users) are clustered by adding edges between similar nodes. GNNs are a Deep Learning (DL) [5] architecture designed to operate as a neural network on graph-structured data. GNNs are extensively used in CF applications [6]. Popular user-item CF models are GC-MC [7], LightGCN [8], NGCF [9] and PinSage [10]. One common implementation approach for CF-based GNNs is to use a Graph Convolution Network (GCN) [11]. GCNs take a similar approach to Convolutional Neural Networks (CNNs) [12], by aggregating features from a node's k-distant neighbors. GCNs are very effective in CF contexts, due to their ability to learn features diffused throughout the network structure, instead of local features only [13]. A GCN can be formally defined as:

$$\mathbf{H}^{(l+1)} = \boldsymbol{\sigma}\left(\sum_{k=0}^{K-1} \tilde{\mathbf{A}}^k \mathbf{H}^{(l)} \mathbf{W}_k^{(l)}\right)$$

Here, $\mathbf{H}^{(l+1)}$ is the feature matrix at the next layer, $\boldsymbol{\sigma}$ is a non-linear activation function, $\tilde{\mathbf{A}}$ is the normalized adjacency matrix of the graph, $\mathbf{H}^{(l)}$ is the feature matrix at the current layer, $\mathbf{W}^{(l)}$ is a learnable weight matrix, and K is the number of filter taps. Choosing an appropriate adjacency matrix $\tilde{\mathbf{A}}$ is critical, as it defines the graph connectivity. The adjacency matrix should be chosen carefully such that it correctly reflects the relationships of the underlying data. One proposed method is to use the sample covariance matrix as $\tilde{\mathbf{A}}$, which forms the basis of coVariance Neural Networks (VNNs) [14].

2.2 Covariance Neural Networks

VNNs are a GNN architecture that operates on the sample covariance matrix $\hat{\mathbf{C}}$ as the graph adjacency matrix $\tilde{\mathbf{A}}$. VNNs have been shown to overcome the two main problems of PCA: failing to capture nonlinearities and instability when the eigenvalues are close [14]. VNNs have also shown better transferability to the underlying data than PCA. This property is critical for CF applications because it reduces the need to retrain the model as the underlying data evolves. VNNs use, just like PCA, the sample covariance matrix defined as

$$\hat{\mathbf{C}}_n \triangleq \frac{1}{n}\sum_{i=1}^{n}(\mathbf{x}_i - \bar{\mathbf{x}})(\mathbf{x}_i - \bar{\mathbf{x}})^{\mathsf{T}}$$

to capture relationships in the data. Here, $\mathbf{x}_i$ is an observation vector, and $\bar{\mathbf{x}}$ is the sample mean vector.

Using the covariance matrix to determine graph relationships can pose problems in the context of CF. Since the covariance matrix is often a dense matrix with high dimensionality (number of users or number of items), meeting memory and computational constraints when computing graph convolutions can be challenging. Additionally, because the covariance matrix captures marginal correlations, it can introduce spurious relationships between users who follow global trends, rather than identifying direct user-to-user similarities. One way to overcome these problems is by using the precision matrix, which measures conditional interdependence between users by taking global relations into account.

2.3 Precision Matrix

The precision matrix is the inverse of the sample covariance matrix defined as

$$\boldsymbol{\Theta} = \hat{\mathbf{C}}^{-1}$$

and measures the conditional independence between variables. That is, for two variables A and B, how much A and B depend on each other given all other variables in the system. Zero entries in the precision matrix mean that two variables are completely independent. This conditional independence eliminates spurious relations caused by two variables that both follow the same global trend, and captures only direct dependency between the variables. Because of this, precision matrices are typically much sparser than their corresponding covariance matrices. In the context of CF, this is particularly useful since it reduces noise in the graph caused by global trends, and reduces overall computational complexity. However, computing the inverse $\hat{\mathbf{C}}^{-1}$ can be numerically challenging, particularly for high-dimensional covariance matrices, which is often the case in CF. Because many user-preference datasets are naturally sparse, the covariance matrix often becomes singular or ill-conditioned, making direct inversion unstable or impossible. Regularization techniques such as Ridge Regularization address this issue by adding a small constant $\lambda > 0$ to the diagonal entries, so the regularized covariance matrix becomes:

$$\hat{\mathbf{C}}_{\text{regularized}} = \hat{\mathbf{C}} + \lambda\mathbf{I}$$

However, introducing a regularization constant λ introduces bias and may lead to incorrect statistical inferences, leading to a larger difference between $\mathbf{\Theta}^{-1}_{\text{regularized}}$ and $\hat{\mathbf{C}}$, where $\mathbf{\Theta}^{-1}_{\text{regularized}}$ is the precision matrix of $\hat{\mathbf{C}}_{\text{regularized}}$. Additionally, by adding the same constant λ to all features uniformly, it can lead to over-regularization or under-regularization of certain features. Since $\lambda > 0$, the precision matrix of the regularized covariance matrix becomes dense, and thus loses the sparsification property that the precision matrix can benefit from. Furthermore, selecting an appropriate regularization constant λ can be challenging, as a too large λ over-regularize certain features, but a too small λ lead to numerical instability. Because of these issues, more advanced regularization techniques are often preferred [15]. One such popular technique is Graphical Lasso [16].

Graphical Lasso

Graphical Least Absolute Shrinkage and Selection Operator (GLASSO) is a regression-based L1-regularization technique for estimating the precision matrix that promotes sparsification and per-feature regularization. Unlike Ridge Regularization, Graphical Lasso learns the optimal estimation of the precision matrix $\mathbf{\Theta}^*$ by using a regression-based model. This allows for learning the optimal regularization feature-wise, instead of adding the same constant to all features. The minimization objective function of Graphical Lasso is defined as:

$$\min_{\mathbf{\Theta}^* \succ 0} \left\{ \underbrace{-\log\det(\mathbf{\Theta}^*)}_{\text{Log-determinant}} + \underbrace{\text{tr}(\hat{\mathbf{C}}\mathbf{\Theta}^*)}_{\text{Data fitting}} + \underbrace{\lambda\|\mathbf{\Theta}^*\|_1}_{\text{Sparsity penalty}} \right\}$$

Here $\mathbf{\Theta}^* \succ 0$ denotes that the estimated precision matrix must be positive definite, which ensures that the resulting covariance matrix estimate remains valid. Each term in the objective function serves a specific purpose:

The **log-determinant term** acts as a barrier function that prevents the eigenvalues of $\mathbf{\Theta}^*$ from becoming too small, which can make the precision matrix ill-conditioned.

The **data fitting term** measures how well the estimated precision matrix fits the observed data. Since ideally $\hat{\mathbf{C}}\mathbf{\Theta}^* \approx \mathbf{I}$, the trace of the matrix (sum of diagonals) should approximate n. Thus, the smaller this term is, the better the fit of the precision matrix

The **sparsity penalty term** controls the sparsity of the estimated precision matrix. The L1 norm (sum of non-diagonal elements) is penalized by λ. Higher values of λ will reduce the L1 norm during the minimization objective, thus encouraging non-diagonal entries to become exactly zero.

Since the objective function is convex, the optimization problem can be solved efficiently using a gradient descent algorithm.

3 Methodology

This section outlines the experimental methodology, encompassing data preparation, model architecture design, training procedures, and evaluation metrics. The conducted experiment aims to evaluate the performance of precision VNNs with different levels of sparsification in the context of CF. A matrix reconstruction approach is used to predict user preferences using a user-user and item-item precision VNN architecture.

3.1 Dataset

The dataset used in the experiment is MovieLens-100K [17], which contains approximately 100,000 movie ratings (ranging from 1 to 5 stars) for 1682 movies by 943 users, with an average of 106 ratings per user. Users and movies are anonymized in the dataset by only revealing their corresponding IDs. For the experiment, only the `user_id`, `movie_id` and `rating` fields are used, excluding genres and timestamps.

3.2 Methods

Before using the MovieLens dataset, the ratings are centered and scaled from values in the range [1, 5] to [-1, 1]. Using these normalized movie ratings, the user-user and item-item sample covariance matrices $\hat{\mathbf{C}}_{\text{user}}$ and $\hat{\mathbf{C}}_{\text{item}}$ are constructed, where undefined values are set to 0. Then the estimated precision matrices $\mathbf{\Theta}^*_{\text{user}}$ and $\mathbf{\Theta}^*_{\text{item}}$ are calculated using Graphical Lasso with hyperparameter λ for controlling the amount of sparsification. Next, the GNN is constructed using $\mathbf{\Theta}^*$ as the graph adjacency matrix $\tilde{\mathbf{A}}$. For the implementation of the GNN, the `SelectionGNN` [18] architecture is used. As the non-linear activation function $\boldsymbol{\sigma}$, ReLU [19] is used. The only feature used in the model is the movie rating given by a user. Before the training process, the data is split into training / validation / testing sets with respective ratios 80% / 10% / 10%. The training process uses a matrix reconstruction approach, in which some ratings are masked and the model is tasked with accurately predicting these masked ratings. The mask ratio α defines the ratio of ratings to be masked randomly in the training data, which will be set to 0 to indicate unknown values. The input is then split into batches of size β. Each batch is used as input $\mathbf{x}$ to the GNN, which outputs predicted batch ratings $\hat{\mathbf{y}}$. Finally, the MSE loss is computed over the masked ratings only of $\hat{\mathbf{y}}$ and true ratings $\mathbf{y}$. The hyperparameters of the model are described in Table 1.

Parameter	Description
λ	Amount of sparsification in the estimated precision matrix
α	Mask ratio of the ratings for matrix reconstruction
β	Batch size
k	Number of filter taps
η	Learning rate

Table 1: Hyperparameters of the model

3.3 Evaluation metrics

As the evaluation metric, the RMSE is used to measure the accuracy of the predicted movie ratings by the model. The RMSE is particularly suitable in the context of recommender systems, as it penalizes larger prediction errors, which can lead to poor recommendations, more heavily. The RMSE is calculated as

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2}$$

where $\hat{y}_i$ represents the predicted rating and y_i the true rating for a sample i. Lower RMSE values indicate better model performance.

4 Results

This section presents the chosen hyperparameters and the outcome of the conducted experiments.

4.1 Hyperparameters

Through experimenting with different hyperparameters, the optimal combination that achieved the lowest RMSE on the dense precision matrix was found to be: $\alpha = 0.3$, $\beta = 5$, $k = 2$, $\eta = 0.01$, and λ for controlling sparsification in the experiments. The parameters α and β are optimized using a search matrix, which can be found in Appendix A.

4.2 Sparsifying the precision matrix

Figure 1 shows the relationship between the Graphical Lasso sparsification level λ, the number of non-zero entries in the estimated precision matrix $\mathbf{\Theta}^*$, and the cost. The result is based on the top 200 users with the most ratings, and the number of iterations for the regression task is set to 100. The cost is defined as the difference between the estimated covariance matrix $\mathbf{\Theta}^{*-1}$ and the sample covariance matrix $\hat{\mathbf{C}}$. As visible in the graph, the number of non-zero entries in the precision matrix drops drastically with higher levels of sparsification. With higher levels of sparsification, the number of non-zero entries converges to only diagonal elements (200 in this case). Furthermore, the cost shows a clear inverse relationship with the number of non-zero entries. Graphical Lasso with $\lambda < 0.0005$ did not converge, as the matrix became too ill-conditioned.

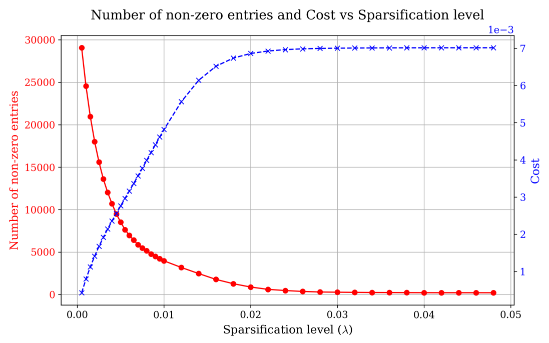

Figure 1: The relation between the Graphical Lasso sparsification level, the number of non-zero entries in the estimated precision matrix, and the cost.

4.3 Performance of the sparsified precision matrix

In Figure 2, the relationship between the Train/Test RMSE and the sparsification level of the precision matrix is plotted. Noticeable is an almost linear increase in the Train RMSE with higher levels of sparsification. The Test RMSE is relatively high with low levels of sparsification, but stabilizes at a minimum in the range $0.0050 < \lambda < 0.0100$, with an RMSE of around 1.010.

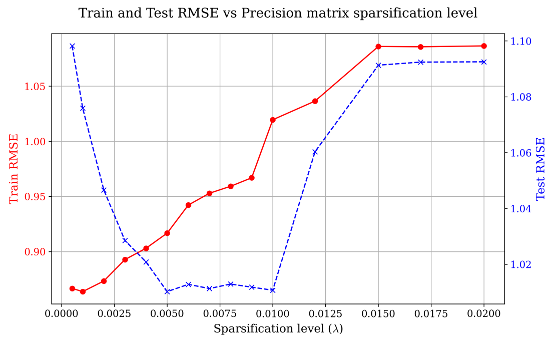

Figure 2: The relation between the Graphical Lasso sparsification level, the Test RMSE, and the Train RMSE of the last epoch.

4.4 Computational complexity

To evaluate computational performance, we measured the training time for 100 epochs at different sparsification levels. In Figure 3 the resulting graph is plotted. Noticeable is that the training time maps almost directly to the number of non-zero entries in the precision matrix, as shown in Figure 1.

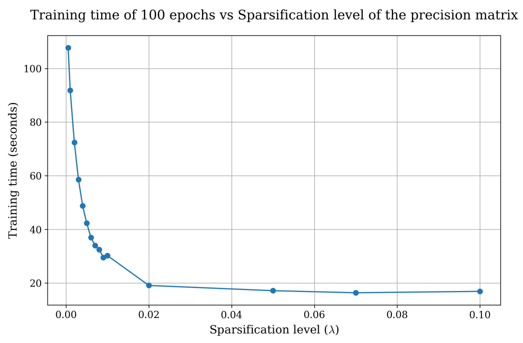

Figure 3: The relation between the training time of 100 epochs, and the sparsification level of the precision matrix.

4.5 Covariance matrix performance

When running the model on the dense sample covariance matrix, the model achieves an RMSE of 0.975. This is 13.7% better than densest precision matrix, which achieved an RMSE of 1.109.

5 Discussion

5.1 Results

As shown in Figure 1, the estimation accuracy of the precision matrix degrades as the level of sparsification increases. This result is expected, as Graphical Lasso has fewer data points from which to learn, making the optimization more challenging. Lower amounts of sparsification thus lead to a more accurate precision matrix, which is better able to capture conditional independence between users. In Figure 2, the increase in Train RMSE confirms this. Since the precision matrix is derived from the training data, more accurate precision matrices (from lower amounts of sparsification) lead to lower RMSE values.

However, a large gap is noticeable between the Test and Train RMSE with lower levels of sparsification. This observation indicates overfitting of the model on the training data. When the precision matrix is dense, and the data it's based on is sparse, the conditional dependencies estimated are mostly noise. A general rule of thumb for an accurate representation of the precision matrix is to use $n \gg p$, where n is the number of samples, and p the number of features [20]. In the case of MovieLens-100K dataset, the number of features per user corresponds to the total number of movies (1682), while the average number of ratings per user (106) is much lower. So in this case $p \gg n$, and thus the relations the precision matrix captures are mostly noise. When the level of sparsification increases, this noise gets filtered out and performance improves. This explains the drop in Test RMSE visible in Figure 2. When the precision matrix becomes too sparse, the model performance degrades again, as it loses too much valuable relationship data between the users.

Since the Sample Covariance Matrix captures marginal relations instead of conditional relations, it's easier to create a more accurate estimation on sparse data. This explains the higher performance of the covariance matrix. The measured RMSE performance of the Covariance Matrix model (0.975) is comparable to other inductive learning models on the MovieLens-100K dataset [21].

When considering computational complexity, the training time significantly reduces as the precision matrix becomes more sparse, visible in Figure 3. The training time corresponds almost directly to the number of non-zero entries in the precision matrix, meaning the training time improves linearly as the number of non-zero entries decreases.

5.2 Limitations

All the experiments are performed on a single dataset which is naturally sparse and high-dimensional. Results can potentially differ when evaluating on different, possibly denser user-item interaction datasets.

Due to computational constraints, the hyperparameter search grid only consists of 2 parameters. The performance of the model could potentially improve by adjusting the other parameters.

6 Conclusions and Future Work

In this paper, we evaluated the performance of sparsified precision VNNs as a graph collaborative filter, using the MovieLens-100K dataset, primarily focusing on the RMSE performance and computational complexity under different levels of sparsification.

Due to the sparse and high-dimensional nature of many collaborative user-preference datasets (such as MovieLens), the estimated precision matrix contained high levels of noise, which resulted in poor performance when using the dense precision matrix. After filtering out some of this noise by using sparsification, the model performance increased and reached a minimum RMSE of 1.010. This is still subpar compared to using the sample covariance matrix as GSO, which achieved an RMSE of 0.975. When measuring computational complexity, the training time reduces significantly with higher levels of sparsification. The results showed a linear relation between the number of non-zero edges in the precision matrix and the training time.

Since this research only evaluated the performance on the MovieLens-100K dataset, for a full understanding of the performance, it's necessary to evaluate the model on different datasets. This dataset should preferably have a higher density than MovieLens-100K, such that the estimated precision matrix becomes more accurate. However, finding such datasets is challenging due to the sparse nature of user-item interaction datasets.

7 Responsible Research

Just like any other machine learning model, GNN recommender systems can pose a risk when incorporating harmful biases in the training process. In the case of the MovieLens-100K, all users and movies are anonymized, keeping only their corresponding numeric ID as identifier. The only feature used for training the model is the movie rating given by a user. This minimizes the risk of explicit demographic bias like age, gender, or ethnicity. However, the implicit rating patterns may reinforce existing user biases and create filter bubbles that limit item diversity. This should be taken into account when using the model in a real-world application, and potentially calibrate it by taking diversity evaluation metrics into account.

All experiments can be reproduced by running the source code published on GitHub [22].

References

[1] Linyuan Lü, Matúš Medo, Chi Ho Yeung, Yi-Cheng Zhang, Zi-Ke Zhang, and Tao Zhou. Recommender systems. 519(1):1–49. Publisher: Elsevier.

[2] Xiaoyuan Su and Taghi M Khoshgoftaar. A survey of collaborative filtering techniques. 2009(1):421425. Publisher: Wiley Online Library.

[3] Hervé Abdi and Lynne J Williams. Principal component analysis. 2(4):433–459. Publisher: Wiley Online Library.

[4] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. 20(1):61–80. Publisher: IEEE.

[5] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. 521(7553):436–444. Publisher: Nature Publishing Group UK London.

[6] Shiwen Wu, Fei Sun, Wentao Zhang, Xu Xie, and Bin Cui. Graph neural networks in recommender systems: a survey. 55(5):1–37. Publisher: ACM New York, NY.

[7] Rianne van den Berg, Thomas N Kipf, and Max Welling. Graph convolutional matrix completion.

[8] Xiangnan He, Kuan Deng, Xiang Wang, Yan Li, Yongdong Zhang, and Meng Wang. Lightgcn: Simplifying and powering graph convolution network for recommendation. In *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval*, pages 639–648.

[9] Xiang Wang, Xiangnan He, Meng Wang, Fuli Feng, and Tat-Seng Chua. Neural graph collaborative filtering. In *Proceedings of the 42nd international ACM SIGIR conference on Research and development in Information Retrieval*, pages 165–174.

[10] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 974–983.

[11] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks.

[12] Zewen Li, Fan Liu, Wenjie Yang, Shouheng Peng, and Jun Zhou. A survey of convolutional neural networks: analysis, applications, and prospects. 33(12):6999–7019. Publisher: IEEE.

[13] Le Wu, Peijie Sun, Richang Hong, Yanjie Fu, Xiting Wang, and Meng Wang. Socialgcn: An efficient graph convolutional network based model for social recommendation.

[14] Saurabh Sihag, Gonzalo Mateos, Corey McMillan, and Alejandro Ribeiro. coVariance neural networks.

[15] Jianqing Fan, Yuan Liao, and Han Liu. An overview on the estimation of large covariance and precision matrices. _eprint: 1504.02995.

[16] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. Sparse inverse covariance estimation with the graphical lasso. 9(3):432–441. Publisher: Oxford University Press.

[17] F Maxwell Harper and Joseph A Konstan. The movielens datasets: History and context. 5(4):1–19. Publisher: Acm New York, NY, USA.

[18] Fernando Gama, Luana Ruiz, and Alejandro Ribeiro. Graph neural networks implementation. URL: https://github.com/alelab-upenn/graph-neural-networks.

[19] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814.

[20] Elizaveta Levina and Roman Vershynin. Partial estimation of covariance matrices. _eprint: 1008.1716.

[21] Seyed Sina Ziaee, Hossein Rahmani, and Mohammad Nazari. MoRGH: movie recommender system using GNNs on heterogeneous graphs. 66(12):7419–7435. Publisher: Springer.

[22] Sparse precision VNN source code. URL: https://github.com/jortboon/sparse-precision-vnn.

A Hyperparameter optimization

$\beta \setminus \alpha$	0.1	0.2	0.3	0.4	0.5
5	1.109	1.105	**1.098**	1.102	1.107
10	1.139	1.135	1.124	1.132	1.137
20	1.193	1.188	1.176	1.185	1.190
50	1.218	1.214	1.202	1.211	1.216

Table 2: Hyperparameter optimization search matrix of the mask ratio α and the batch size β (RMSE test performance on the dense precision matrix with $\lambda = 0.0005$).

B Use of AI

A Large Language Model (ChatGPT) is used to give relevant context and improve understanding of topics related to this research. All code, training procedures, and the paper itself, are created from scratch without any assistance of a LLM.