Low-Rank Matrix Completion Using Graph Neural Network

Luong Trung Nguyen  
Dept. of Electrical and Computer Engineering  
Seoul National University, Seoul, Korea  
ltnghuyen@islab.snu.ac.kr

Byongyo Shim  
Dept. of Electrical and Computer Engineering  
Seoul National University, Seoul, Korea  
bshim@islab.snu.ac.kr

Abstract—In this paper, we propose the graph neural network (GNN)-based matrix completion technique to reconstruct the desired low-rank matrix by exploiting the underlying graph structure of the matrix. The proposed approach, referred to as GNN-based low-rank matrix completion (GNN-LRMC), combines the GNN and the neural-network weight update mechanism. The GNN is used to extract the node vectors of the graph using a modified convolution operation. Empirical simulations validate the reconstruction performance of GNN-LRMC in synthetic and Netflix datasets.

Index Terms—low-rank matrix completion, convolutional neural network, graph neural network

I. INTRODUCTION

In recent years, low-rank matrix completion (LRMC) has been found its way in many applications such as recommendation system, IoT localization, image recovering, wireless channel estimation [1], to name just a few [2]. Basic premise of LRMC is that the unknown entries of a matrix can be reconstructed using a small subset of observed entries when the matrix has the low-rank structure. It has been shown that under some suitable conditions, a low-rank matrix can be recovered accurately with overwhelming probability with a small number of observed entries [2].

In many practical situations, such as recommendation systems, social networks, and computer graphics, a desired low-rank matrix can be represented using a graph structure. For example, in the recommendation systems, a rating matrix $M$ containing user’s feedback on the purchased items can be represented as graphs of users and products. In the user graph, an edge connecting two user nodes expresses the correlation between two users’ favor (see Fig. 1). In a similar way, in the product graph, an edge represents the correlation between two products. In fact, this graph structure plays a key role in reconstructing unknown entries of $M$. As a simple example, consider the $4 \times 4$ rating matrix $M$ in Fig. 1. When the rank of $M$ is 2, then in general at least two known entries should be known in each column to reconstruct the matrix$^1$. Interestingly, in the graph-based LRMC, we can recover $M$ even with only one known entry in its second column using the local connectivity of the graph structure.

To be specific, we decompose $M = UV^T$ and then map the column vectors of $V^T$ to the nodes of the column graph (see Fig. 1). In doing so, we can express the unknown node vector $v_2$ using the local connectivity of the graph. That is,

$$v_2 = \begin{bmatrix} x \\ y \end{bmatrix} = \alpha \begin{bmatrix} 0 \\ 3 \end{bmatrix} + \beta \begin{bmatrix} 0 \\ 3 \end{bmatrix},$$  

where $\alpha$ and $\beta$ are the linear coefficients. It is clear from (1) that $x = 0$. Also, it is easy to see that $y = 2$ using $M(2, 2) = u_2^T v_2$ ($u_i$ is the $i$-th row of $U$). This example is obviously simple, but the fundamental principle to recover unknown entries of a large dimensional matrix using the graph structure is not much different from this and the local connectivity in a graph plays a pivotal role in recovering the matrix. In recent years, graph neural network (GNN) has been employed to identify unknown vectors (node vectors) in $U$ or $V^T$ [3]–[5]. In this approach, convolutional layers are used to extract features in the graph (i.e., vector-valued data at graph nodes).

---

\[^1\]Since the rank is 2, each column should be expressed as a linear combination with the other two columns. Then at least two known entries are used to reconstruct the linear coefficients.
An aim of this paper is to propose the GNN-based LRMC technique, referred to as GNN-LRMC, which elegantly combines the GNN and a neural-network weight update mechanism. Key idea of the proposed approach is to train two parameter matrices $U_o$ and $V_o$ such that they learn the local connectivity of the graph. The edge weights of the row and column graphs are defined as the entries of the correlation matrices $C_{uu}$ and $C_{vv}$ of $U_o$ and $V_o$, respectively. Use of fully connected layer together with proper activation (slicing) function allows us to eliminate small entries in $C_{uu}$ and $C_{vv}$, resulting in the removal of unnecessary connections in the graph. In the proposed approach, the GNN is used to update the node vectors of the graph by performing the convolution over the graph. In the training process, we update the network parameters (weights and biases) using the supervised learning. From the empirical simulations using synthetic and Netflix datasets, we show that the proposed GNN-LRMC outperforms the conventional techniques, resulting in 50% improvement of the reconstruction error.

II. PROPOSED GNN-LRMC

In this section, we first discuss the graph model and then explain the proposed GNN-LRMC.

A. Graph Model

Let $M = UV^T \in \mathbb{R}^{n_1 \times n_2}$ be the matrix factorization of rank-$k$ matrix $M$ using $U \in \mathbb{R}^{n_1 \times k}$ and $V \in \mathbb{R}^{n_2 \times k}$. Then, $U$ and $V$ can be mapped to the row and column graphs $G_r = (V_r, E_r, W_r)$ and $G_c = (V_c, E_c, W_c)$ of $M$, respectively. Note that $V_r$ and $V_c$ are vertex sets, $E_r$ and $E_c$ are edge sets, and $W_r$ and $W_c$ are the weight matrices (e.g., the entry $w_{ij, j_3}$ is the weight coefficient of the edge connecting two nodes $u_{j_1}$ and $u_{j_3}$ (see Fig. 2)). For example, the row vectors of $U$ are mapped to the nodes of $G_r$. The edge weights of $G_r$ are computed based on the correlation between the row vectors of $U$.

B. Neural-Network Weight Computation

In the proposed GNN-LRMC, we compute the weight matrices $W_r$ and $W_c$ using the neural-network weight update mechanism (see Fig. 2). To be specific, we first initialize the trainable matrices $U_o$ and $V_o$. The correlation matrices of $U_o$ and $V_o$ are defined as

$$C_{uu} = -\frac{1}{\beta_u}(D_u - \alpha_u 11^T) \quad \text{and} \quad C_{vv} = -\frac{1}{\beta_v}(D_v - \alpha_v 11^T),$$

where $D_u$ is the distance matrix of $U_o$ [6], [7], $I = \begin{bmatrix} 1 & \cdots & 1 \end{bmatrix}^T$, $\alpha_u = \frac{1}{n_1} \sum_{i,j} |D_u|_{ij}$, and $\beta_u = \max_{i,j} |D_u|_{ij}$. Note that $\alpha_u$, $\alpha_v$, $\beta_u$, and $\beta_v$ are used to ensure that $C_{uu}$ and $C_{vv}$ are unbiased and normalized. Then, $W_r$ and $W_c$ can be computed using a fully connected neural network. That is,

$$W_r = \sigma(C_{uu} + b_u 1^T) \quad \text{and} \quad W_c = \sigma(C_{vv} + b_v 1^T),$$

where $\sigma$ is the rectified linear unit (ReLU) activation function and $b_u$ and $b_v$ are the bias vectors. Note that the use of the ReLU function together with a proper threshold $b_u$ and $b_v$ allows us to truncate the small weights, which helps to avoid overfitting and also reduce the model complexity. Note also that $U_o$ and $V_o$ are updated using the back propagation in the training process to obtain the optimal weight matrices $W_r$ and $W_c$ (see Fig. 2a).

2When the weight is zero, there is no connection between two nodes in the graph.
C. GNN

Taking $U_o$ and $V_o$ as the input and using the graph-based convolution, GNN generate estimates $\hat{U}$ and $\hat{V}$ of $U$ and $V$ (see Fig. 2a). Here, the $i$-th row vector $\hat{u}_i$ of $\hat{U}$ ($\hat{v}_j$ of $\hat{V}$) is the updated node vector at the $i$-th node in the row graph $G_r$ (the $j$-th node in the column graph $G_c$) obtained from the convolution over the graph. That is, $\hat{u}_i$ and $\hat{v}_j$ can be expressed as

$$\hat{u}_i = f(u_i, N_q(u_i))$$  \hspace{1cm} (4)

$$\hat{v}_j = f(v_j, N_q(v_j)),$$  \hspace{1cm} (5)

where $u_i$ is the $i$-th row of $U_o$, $v_j$ is the $j$-th row of $V_o$, and $f$ is the graph-based convolution. Note that $N_q(u_i)$ is defined as the $q$-hop neighbors of $u_i$, defined as the nodes with the shortest path to $u_i$ not being greater than $q$ (see Fig. 2b). For example, $N_1(u_i)$ is the set of adjacent nodes of $u_i$.

In a graph-based convolution, $\hat{U}$ and $\hat{V}$ are computed as [3]

$$\hat{U} = f(U_o) = \sum_{t=0}^{q-1} \theta_t R_r^t U_o$$ \hspace{1cm} (6)

$$\hat{V} = f(V_o) = \sum_{t=0}^{q-1} \delta_t R_c^t V_o,$$ \hspace{1cm} (7)

where $\theta_t$ and $\delta_t$ are the convolutional filter parameters, $R_r$ and $R_c$ are the symmetric normalized Laplacian matrices$^3$ of $G_r$ and $G_c$, respectively. Note that the convolution operation $f$ is localized in (6) and (7), meaning that it is performed on the local area of each node in the graph$^4$ (see Fig. 2b).

In the proposed approach, we modify the convolution operation $f$ based on a generalized Laplacian matrix to stabilize the training process. Note that $R_r$ and $R_c$ in (6) and (7) do not have DC component and thus might not handle constant signals (i.e., the node vector that does not need to be adjusted using the values of adjacent nodes of the node) [8]. To overcome this, we use a generalized Laplacian $R_r$ defined as

$$\bar{R}_r = (1 + \tau_r)I - \tau_r D_r^{-1} W_r$$ \hspace{1cm} (8)

$$\bar{R}_c = (1 + \tau_c)I - \tau_c D_c^{-1} W_c,$$ \hspace{1cm} (9)

where $\tau_r$ and $\tau_c$ are tuning parameters, $D_r = \text{diag}(W_r 1)$, $D_c = \text{diag}(W_c 1)$, and $I = \left[ \begin{array}{cccc} 1 & 1 & \cdots & 1 \end{array} \right]^T$. The outputs $\hat{U}$ and $\hat{V}$ of the modified convolution are given, respectively, as

$$\hat{U} = f(U_o) = \sum_{t=0}^{q-1} \theta_t \bar{R}_r^t U_o$$ \hspace{1cm} (10)

$$\hat{V} = f(V_o) = \sum_{t=0}^{q-1} \delta_t \bar{R}_c^t V_o.$$ \hspace{1cm} (11)

Note that the filter parameters $\theta_t$ and $\delta_t$ are updated using the back propagation in the training process.

$^3$In a graph, the Laplacian matrix is a discrete analog of the Laplacian operator.

$^4$It can be shown that the local area consists of the $p$-hop neighbors of the node.

D. Output Model

In GNN-LRMC, output model is a mapping $\phi$ between the extracted feature $(\hat{U}, \hat{V})$ and the reconstructed low-rank matrix $\hat{M}$:

$$\hat{M} = \phi(\hat{U} \hat{V}^T).$$ \hspace{1cm} (12)

In our study, to avoid a high model complexity and also reduce the memory size, we use a simplified fully connected layer. In this model, the reconstructed matrix $\hat{M}$ is expressed as $\hat{M} = \phi(\bar{U} \bar{V}^T) = \sigma(s \bar{U} \bar{V}^T + b1_1^T)$ where $s$ is a scale parameter and $b$ is an offset constant.

E. Training Cost Function

In order to minimize the reconstruction error, Frobenius norm based cost function is widely used [2]. Let $\Omega$ be the set of indices of known entries, then $P_{\Omega}$ is the sampling operator defined as

$$[P_{\Omega}(A)]_{ij} = \begin{cases} a_{ij} & \text{if } (i,j) \in \Omega \\ 0 & \text{otherwise} \end{cases}.$$ \hspace{1cm} (13)

Our training cost function is given by

$$l(\hat{U}, \hat{V}) = \sum_{(i,j) \notin \Omega} w_{r,ij} ||\hat{u}_i - \hat{u}_j||_2 + \sum_{(i,j) \in \Omega} w_{c,ij} ||\hat{v}_i - \hat{v}_j||_2$$

$$+ \rho ||P_{\Omega}(\hat{M}) - P_{\Omega}(M)||_F,$$ \hspace{1cm} (14)

where $\rho$ is the regularization parameter and $w_{r,ij}$ and $w_{c,ij}$ are the entries of $W_r$ and $W_c$, respectively. Note that we use the additional $\ell_2$-norm terms $||\hat{u}_i - \hat{u}_j||_2$ and $||\hat{v}_i - \hat{v}_j||_2$ to promote the correlation among adjacent node vectors. To initialize the graph weight coefficients, we pre-train the network using the cost function:

$$\kappa(U_o, V_o) = \sum_{ij} |w_{r,ij} - w_{ro,ij}| + |w_{c,ij} - w_{co,ij}|,$$ \hspace{1cm} (15)

where $w_{ro,ij}$ and $w_{co,ij}$ are the initialized values of $w_{r,ij}$ and $w_{c,ij}$, respectively.

In Table I, we summarize the proposed GNN-LRMC algorithm.

<table>
<thead>
<tr>
<th>TABLE I</th>
</tr>
</thead>
<tbody>
<tr>
<td>GNN-LRMC ALGORITHM</td>
</tr>
<tr>
<td>Input</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Initialize</td>
</tr>
<tr>
<td>weight matrices $W_{ro}$ and $W_{co}$</td>
</tr>
<tr>
<td>While</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>End</td>
</tr>
<tr>
<td>Output</td>
</tr>
</tbody>
</table>
In this paper, we proposed a deep-learning based LRMC technique called GNN-LRMC that combines the GNN and the neural-network weight update mechanism. Empirical study shows our proposed GNN-LRMC can significantly improve the accuracy of the low-rank matrix reconstruction and outperform conventional techniques. We believe that our approach can be easily extended to various scenarios where the desired low-rank matrix has a graph structure such as computer graphics, social network analysis.

**ACKNOWLEDGMENT**

This work was supported by Samsung Research Funding & Incubation Center for Future Technology of Samsung Electronics under Project Number (SRFC-IT1901-17).

**REFERENCES**


