# CS 782: ADVANCED ML HOMEWORK 2

Gaurab Pokharel, FCD Samuel

Email:{gpokhare, sfrank22}@gmu.edu

**Statement of Contribution**: Gaurab and Sam collaborated on this homework. They worked on the coding aspect separately and came together to combine their work to one coherent document. They divided the job of the write-up equally amongst themselves.

> The matrix completion problem involves inferring missing or unobserved entries in a matrix based on the known entries. It finds applications in various domains such as recommender systems, collaborative filtering, and image inpainting. Mathematically, the problem can be formulated as an optimization task, where the objective is to reconstruct the entire matrix using a subset of observed entries. Several algorithms, including low-rank matrix factorization, convex optimization, and iterative methods like gradient descent, have been proposed to address the matrix completion problem.

Mathematically, the problem statement can be formulated as an optimization problem, as shown in Equation (1), where the objective is to minimize the squared error between the reconstructed matrix and the observed entries.

$$\min_{U \in \mathbb{R}^{d \times d}} \frac{1}{2} \sum_{(i,j) \in \Omega} ((UU^T)_{ij} - Y_{ij})^2, \tag{1}$$

where $Y \in \mathbb{R}^{d \times d}$ is a symmetric and positive semi-definite low rank target matrix, $\Omega$ is a set of indices for observed entries, and $U \in \mathbb{R}^{d \times d}$ is the matrix we aim to compute.. The matrix completion problem wants to reconstruct all entries in $Y$ with matrix $U$. This problem is particularly challenging when dealing with large, high-dimensional matrices with missing entries.

To tackle this problem, we employ a stochastic gradient descent algorithm, a widely used optimization technique, to iteratively update the matrix $U$ based on the observed indices and the error between the reconstructed and observed entries. This algorithm gradually refines the matrix $U$ until convergence.

```python
import numpy as np
def matrix_completion(Y, Omega, alpha, eta, num_iterations):
    d = Y.shape[0]
    U = alpha * np.eye(d)  # Initialize U with small alpha

    for _ in tqdm(range(num_iterations)):
        grad = np.zeros_like(U)

        for i, j in Omega:
            grad += ((np.dot(U, U.T)[i, j] - Y[i, j]) * (U[:, j].reshape(-1, 1) + U[:,
            ↪  i].reshape(-1, 1)))

        U -= eta * grad  # Update U using gradient descent

    return U
```

As you can see in the code above, the gradient descent algorithm takes in as input the target matrix $Y$, the observation indices $\Omega$, alpha $\alpha$ which is the small value to make $U$ that we initialize close to 0, learning rate $\eta$, and the maximum

number of iterations as input. Then, we calculate the gradient in line 10 of the code based on the observation indices and make updates to $U$ (standard gradient descent algorithm).

In our experiment, we systematically evaluate the performance of the matrix completion algorithm across different matrix dimensions $d \in \{50, 100, 150, 200\}$. For each dimension, we generate a low-rank ground truth matrix $Y$ and randomly sample observed indices $\Omega$. By running the matrix completion algorithm, we obtain the reconstructed matrix $U$, from which we compute metrics such as the nuclear norm and the rank.

The nuclear norm of a matrix is a convex relaxation of its rank, defined as the sum of its singular values. Mathematically, the nuclear norm $\|\cdot\|_*$ of a matrix $A$ is given by:

$$\|A\|_* = \sum_{i=1}^{\min(m,n)} \sigma_i(A)$$

where $\sigma_i(A)$ represents the $i$-th singular value of $A$, and $m$ and $n$ denote the number of rows and columns of $A$ respectively. The nuclear norm serves as a surrogate measure for the rank of the matrix and is commonly used as a regularization term in low-rank matrix recovery problems, including matrix completion and robust principal component analysis.

```
# Set the parameters
d_values = [50, 100, 150, 200]  # Matrix dimensions to test
r = 5    # Rank of the ground truth matrix
num_observed = 1000  # Number of observed entries
alpha = 1e-3  # Small initialization parameter
eta = 1e-3    # Small step size
num_iterations = 10000  # Number of iterations

nuclear_norms = []
reconstructed_ranks = []

for d in d_values:
    # Generate a low-rank ground truth matrix Y
    L = np.random.randn(d, r)
    Y = np.dot(L, L.T)

    # Randomly sample observed indices Omega
    Omega = [(i, j) for i in range(d) for j in range(i, d)]
    np.random.shuffle(Omega)
    Omega = Omega[:num_observed]

    # Run matrix completion using gradient descent
    U = matrix_completion(Y, Omega, alpha, eta, num_iterations)

    # Calculate the nuclear norm and rank of the reconstructed matrix
    reconstructed_matrix = np.dot(U, U.T)
    nuclear_norm = np.sum(np.linalg.svd(reconstructed_matrix, compute_uv=False))
    reconstructed_rank = np.linalg.matrix_rank(reconstructed_matrix)

    nuclear_norms.append(nuclear_norm)
    reconstructed_ranks.append(reconstructed_rank)
```

The nuclear norm serves as a measure of the complexity or 'size' of the reconstructed matrix, while the rank indicates its intrinsic dimensionality. Our experimental results, visualized in Figures 1 and 2, provide insights into the behavior of the algorithm across varying matrix dimensions. We plot our results using the following code which makes the plots of the nuclear norm vs dimension, and the plot for the reconstructed rank vs dimension.

```python
# Plot dimension vs nuclear norm
plt.figure(figsize=(8, 6))
plt.plot(d_values, nuclear_norms, marker='o')
plt.xlabel('Matrix Dimension')
plt.grid(color = 'grey', linestyle = '--', linewidth = 0.5, alpha=0.3)
plt.xticks(np.arange(50, 200, 10))
plt.yticks(np.arange(20, 350, 20))
plt.ylabel('Nuclear Norm')
plt.title('Dimension vs Nuclear Norm')
plt.tight_layout()
plt.show()


# Plot dimension vs reconstructed rank
plt.figure(figsize=(8, 6))
plt.plot(d_values, reconstructed_ranks, marker='o')
plt.xlabel('Matrix Dimension')
plt.ylabel('Reconstructed Rank')
plt.title('Dimension vs Reconstructed Rank')
plt.grid(color = 'grey', linestyle = '--', linewidth = 0.5, alpha=0.3)
plt.xticks(np.arange(50, 200, 10))
plt.yticks(np.arange(40, 200, 10))
plt.tight_layout()
plt.show()
```

The results of our experiments are shown in the figures below:

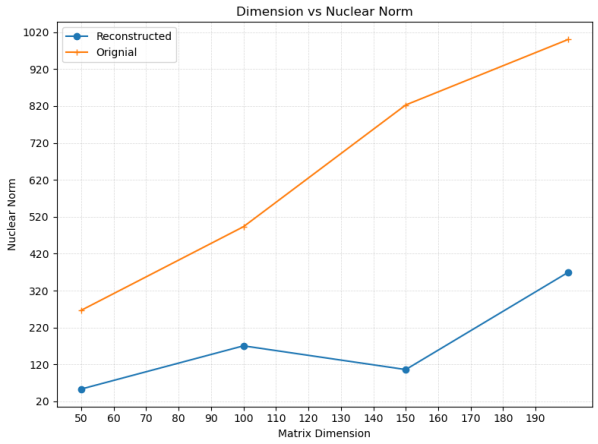


Figure 1: The dimension of the original matrix versus the nuclear norm of the reconstructed matrix and the original matrix

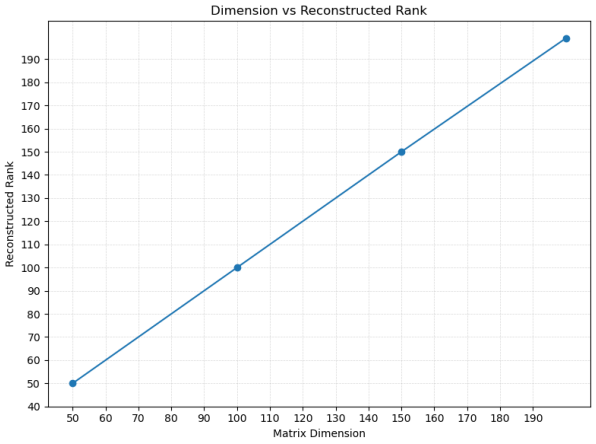


Figure 2: The dimension of the original matrix versus the rank of the reconstructed matrix

Figure 1 illustrates the relationship between the matrix dimension and the nuclear norm of the reconstructed matrix along with the original matrix. It reveals that the nuclear norm tends to increase with the dimensionality of the original matrix, reflecting the overall 'size' of the reconstruction. That being said, the norm of the reconstruction is still less than the norm of the original matrix. This can be an indication that the solution has converged to the min-norm solution for matrix reconstruction.

On the other hand, Figure 2 depicts the relationship between the matrix dimension and the reconstructed rank. Interestingly, we observe that the reconstructed rank does not match the rank of the ground truth matrix, which was set to 5 in our experiment. This discrepancy suggests that the algorithm may not fully capture the intrinsic structure of the original matrix.

We attribute this phenomenon to the inherent limitations of the gradient descent approach, which focuses solely on minimizing the error on the observed indices. As a result, the reconstructed matrix may lack linear independence among its columns, leading to a higher-than-expected rank.

To address this issue, future research could explore incorporating additional constraints or regularization techniques to encourage low-rank reconstructions. Alternatively, initializing the matrix $U$ with a predefined rank estimate could potentially improve the algorithm's performance in preserving the rank of the original matrix.

In summary, our study sheds light on the challenges and opportunities in matrix completion, highlighting the importance of considering both the observed data and the underlying structure of the matrix during reconstruction.