Tensor Algebra
University Federal do Ceará
PPGETI - 2020.2
Saulo Mendes de Melo

Homework 5

Kronecker Product Singular Value Decomposition (KPSVD)

1 Problem 1

On practice 04 we implement the LS-KF(Least Square Kronecker Factorization) algorithm, now we will go to implement a generalization of that. Let

$$\mathbf{X} = \begin{pmatrix} \mathbf{X}_{1,1} & \dots & \mathbf{X}_{1,c_2} \\ \vdots & \ddots & \vdots \\ \mathbf{X}_{r_2,1} & \dots & \mathbf{X}_{r_2,c_2} \end{pmatrix}, \ \mathbf{X}_{i_2,j_2} \in \mathbb{R}^{r_1 \times c_1}.$$

Implement the KPSVD for the matrix $\mathbf{X}$ by computing $\sigma_k, \ \mathbf{U}_k, \ \mathbf{V}_k$ such that

$$\mathbf{X} = \sum_{k=1}^{r_{KP}} \sigma_k \mathbf{U}_k \otimes \mathbf{V}_k$$

Results

The first step of the Solution is to implement a method for transforming a given matrix $\mathbf{X} \in \mathbb{C}^{MP \times NQ}$ such that $\mathbf{X} = \mathbf{A} \otimes \mathbf{B}$ into the reshaped format $\tilde{\mathbf{X}} \in \mathbb{C}^{PQ \times MN}$ such that $\mathbf{X} = vec(\mathbf{B})vec(\mathbf{A})^T$. This matrix format is rank-1. Listing 1 contains this method.

```python
def X_tilde_reshape(X, h_splits, v_splits):
    '''
    This method reshapes an A_kron_B matrix
    into a vec(A)vec(B)^T shaped matrix
    ________________________________________
    Inputs:
        X: Matrix to be reshaped
        h_splits: Number of horizontal splits
        v_splits: Number of vertical splits
    ________________________________________
    Outputs:
        X_til: X reshaped as vec(A)vec(B)^T
    '''

    X_til = []

    for n_matrix in np.hsplit(X, h_splits):
        for a_B in np.vsplit(n_matrix, v_splits):
            X_til.append(a_B.T.reshape(1, -1)[0])

    X_til = np.array(X_til).T

    return X_til
```

Listing 1: $\mathbf{X} \rightarrow \tilde{\mathbf{X}}$ method

This method takes in an input matrix X, the number of horizontal splits and the number of vertical splits. These split numbers refer to the amount of "block" matrices that $\mathbf{X}$ will get divided into, so that each of these blocks gets vectorized in the final $\tilde{\mathbf{X}}$ form. This is essentially the $(r_2,\ c_2)$ shape of $\mathbf{A}$ when $\mathbf{X} = \mathbf{A} \otimes \mathbf{B}$.

```
def KPSVD(X, r1, c1, k):
    '''
    Calculates the Kronecker Product
    Singular Value Decompositon

    ___________________________________________
    Inputs:
        X: Input matrix
        r1: Number of rows of KP block matrix
        c1: Number of columns of KP block matrix
        k: Desired rank of output
    ___________________________________________
    Outputs:
        result: Resulting matrix made up of
            Sigma*kron(U, Vh)

    '''
    v_splits = int(X.shape[0]/r1)
    h_splits = int(X.shape[1]/c1)
    X_til = X_tilde_reshape(X, h_splits, v_splits)

    U, S, Vh = svd(X_til)

    R1_matrices = []

    for i in range(k):
        R1_matrices.append(
            S[i]*kron(Vh[i, :].reshape(h_splits, v_splits).T, U[:, i].reshape(c1, r1).T)
        )

    result = np.sum(np.array(R1_matrices), axis=0)

    return result
```

Listing 2: Calculation of KPSVD

The KPSVD method consists of calculating the SVD of $\tilde{\mathbf{X}}$

$$\tilde{\mathbf{X}} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^H = \sum_{k=1}^{r_{KP}} \sigma_k \mathbf{u}_k \mathbf{v}_k^H$$

Then, for $k \in \{1,\ \ldots,\ r_{kp}\}$, $\mathbf{U}_k$ and $\mathbf{V}_k$ can be defined as

$$\mathbf{U}_k = unvec_{M\times N}\{v_k^T\}$$
$$\mathbf{V}_k = unvec_{P\times Q}\{u_k\}$$

The code in Listing 2 applies this method. It takes an input matrix $\mathbf{X}$, a shape $(r_1,\ c_1)$ corresponding to the size of the "block" matrix in $\mathbf{X}$ (size of $\mathbf{B}$ in $\mathbf{X} = \mathbf{A} \otimes \mathbf{B}$), and k which is the nearest rank-r such that $r \leq r_{kp}$. The result returned is the summation $\sigma_1 \mathbf{U}_1 \otimes \mathbf{V}_1 + \sigma_2 \mathbf{U}_2 \otimes \mathbf{V}_2 + \cdots + \sigma_k \mathbf{U}_k \otimes \mathbf{V}_k$.

```
np.random.seed(0) # set a seed so that random matrices fixed

r2, c2 = 3, 3
r1, c1 = 3, 3

A = randn(r2, c2)
B = randn(r1, c1)

X = kron(A, B)

norm(X - KPSVD(X, r1, c1, 1))
```

Listing 3: Test for KPSVD

The code in Listing 3 defines a simple test for the KPSVD function using $\mathbf{X} \in \mathbb{R}^{9\times 9}$ such that $\mathbf{X} = \mathbf{A} \otimes \mathbf{B}$, being $\mathbf{A} \in \mathbb{R}^{3\times 3}$ and $\mathbf{B} \in \mathbb{R}^{3\times 3}$. Since $\tilde{\mathbf{X}}$ is rank-1 for a Kronecker Product matrix $\mathbf{X}$, considering $r = 1$ for the KPSVD should return a matrix equals to $\mathbf{X}$. The $norm()$ indicated in the code returns 1.979e-15, which is sufficiently close to zero and such a small error is most likely due to data type castings or irrelevant numerical errors.

2 Problem 2

At the above problem, set $r_1 = r_2 = c_1 = c_2 = 3$ and choose $\mathbf{A}_{i,j} = rand(r_1, c_1)$, $1 \leq i \leq r_2$, $1 \leq j \leq c_2$. Then compute the KPSVD and r_{KP} of $\mathbf{A}$ by using your KPSVD prototype function. Consider $r \leq r_{KP}$. Compute the nearest rank-r for the matrix $\mathbf{A}$.

Results

The code in Listing 4 creates $\mathbf{A} \in \mathbb{R}^{9\times 9}$. The value of r_{KP} calculated for $\tilde{\mathbf{X}}$ is 9, which means that it is full-rank, and the nearest rank-r matrix for $\mathbf{A}$ is in $\mathbb{R}^{9\times 9}$. This is behavior is much different than with a Kronecker Product matrix, which is expected, since a random number matrix is not necessarily rank-1 or even low-rank.

```
1  np.random.seed(0) # set a seed so that random matrices fixed
2
3  r2, c2 = 3, 3
4  r1, c1 = 3, 3
5
6  A = randn(r1*r2, c1*c2)
7  R_kp = matrix_rank(X_tilde_reshape(A, c2, r2))
8
9  errors = []
10 for i in range(1, R_kp):
11     errors.append(norm(A - KPSVD(A, r1, c1, i))**2)
```

Listing 4: Problem 2 code

The loop in line 10 performs a squared error calculation for each rank-r decomposition of $\mathbf{A}$. The results can be seen in graph 1. Expectedly, there is a large error for a rank-1 decomposition of a full-rank matrix, and the error steadily decreases as higher rank matrices are used.

Figure 1: Squared Error vs nearest rank-r matrix