BMF: Block Matrix Approach to Factorization of Large Scale Data

A Preprint

Prasad G Bhavana
School of Computer and Information Sciences
University of Hyderabad
India

Vineet C Nair
School and Computer and Information Sciences
University of Hyderabad
India

January 3, 2019

Abstract

Matrix Factorization (MF) on large scale matrices is computationally as well as memory intensive task. Alternative convergence techniques are needed when the size of the input matrix is higher than the available memory on a Central Processing Unit (CPU) and Graphical Processing Unit (GPU). While alternating least squares (ALS) convergence on CPU could take forever, loading all the required matrices on to GPU memory may not be possible when the dimensions are significantly higher. Hence we introduce a novel technique that is based on considering the entire data into a block matrix and relies on factorization at a block level.

Keywords Matrix Factorization, Block Matrix Factorization, GPU Accelerated Matrix Factorization, Matrix Factorization for large datasets

1 Introduction

Matrix factorization is a well known technique for various machine learning problems ranging from recommender systems to text mining. The simplicity of the technique is based on deriving underlying latent factors leading to a behavior by utilization of large data of observed behaviors.

Computation of MF on a CPU is viable only for smaller matrices as the number of computations arithmetically increase with each extra row and/or column and with each additional latent feature derived. The number of computations for MF of a data matrix $X \in \mathbb{R}^{n \times m}$ into latent feature matrices $U \in \mathbb{R}^{n \times k}$, $V \in \mathbb{R}^{m \times k}$ can be arrived as:

$$\approx (n^2 \times m^2 \times a + n \times m \times k \times b + n \times m \times c) \times N$$

for some constants $a$, $b$ and $c$ and for $N$ number of iterations. Hence the time complexity $\approx O(n^4)$

As $n, m \rightarrow \infty$ and as $N, k$ remain constants, the cost of computation $\approx n^2 \times m^2$.

i.e as $n, m \rightarrow 10^6$, the number of computations $\approx 10^{24}$. To put this into perspective, a movie data set of six years with 10 million users and 1 million movies nearly takes 60 days to converge on an Intel(R) Xeon(R) CPU E5-2640 v3 @ 2.60GHz for 30 latent features and 2000 iterations.

MF implemented on a GPU gives substantial performance gain as the number of computations are nearly divided by number of GPU cores (with each core sharing load in parallel computation) leveraged. However, the amount of memory available on a typical GPU is limited and for large data sets it may not be possible to compute MF at once with all data transferred into GPU cache. The memory required for a data matrix can be arrived as:

$$\approx (n \times m + n \times k + k \times m) \times c$$

for some constant $c$ bytes taken to represent each element. Hence the space complexity $\approx O(n^2)$
Considering $k$ as constant, and as $n, m \to 10^6$, the number of memory units required $\approx 10^{12} \times c$.

The challenge of factorizing extremely large data matrix can be achieved by loading part of the data into GPU and continuing to transfer data back and forth between GPU and persistent storage at regular intervals to ensure all data elements are considered for convergence. This is a cumbersome approach that also consumes additional time for frequent data transfers. The movie data set of six years with 10 million users and 1 million items, represented in dense format, consumes $\approx 40,000$ Giga Bytes (GB) of memory. Assuming 8 GB of memory availability on a given GPU with 2 Giga bits/sec transfer rate requires a total of 5000 back and forth data transfers that nearly consume 200 hours of time for data transfer alone.

[1] proposed an alternative approach that uses Stochastic Gradient Dissent (SGD) based convergence instead of ALS with approximately similar convergence and also optimization of memory to reduce the time of data transfer into GPU for computation. While this approach provides computational gain, the approach still requires frequent data transfers for large data sets that can not be loaded on to CPU, GPU.

[2] proposed divide and conquer approach for parallelism of MF by treating factorization of each sub-matrix as a sub-problem that resulted in noisy factorization. Similarly [3] proposed localized factorization for recommendation on a block diagonal matrix.

[4] proposed technique that treats the data matrix as a combination of sub-matrices for LU factorization technique and concluded the factorization outcome as incomplete. Similarly [5] proposed factorization for block matrices.

[6] proposed a block kernel approach to factorization for applications in face recognition.

[7] proposed GPU accelerated Non-Negative Matrix Factorization (NNMF) for CUDA (Compute Unified Device Architecture) capable hardware. Similarly [8] proposed NNMF with GPU acceleration for text mining purposes.

Introduced below is a novel approach that considers any given matrix into a block matrix with each sub matrix independently converged, preferably on a GPU, the resultant latent feature sub-matrices used for factorization of other relevant sub-matrices and the overall process is repeated till convergence equivalent to regular MF.

### 2 The Approach

Let $X \in \mathbb{R}^{(n \times m)}$ be a matrix with $n$ users and $m$ observations per user. Typically, in a regular MF approach, the data matrix $X$ can be approximated as $X \approx UV^T$; where each column of the user latent factor matrix $U \in \mathbb{R}^{(n \times k)}$, for some given $k$ dimensional latent feature vectors derived for each user from the given observations. Similarly, each column of item latent factor matrix $V \in \mathbb{R}^{(m \times k)}$ dimensions of latent feature vector derived for each item from the given observations.

The optimization problem of MF can be represented as:

$$\min_{U,V} = \sum_{x_{ij}} (x_{ij} - u_i^T v_j)^2 + \lambda \sum_u n_u ||u_i||^2 + \sum_v n_v ||v_j||^2$$

and the update equation for $i^{th}$ row of $U$ is given as:

$$\sum_{x_{ij}} (v_j v_j^T + \lambda I). u_i = U^T.X_i^*$$

similarly the update equation for $j^{th}$ column of $V$ is given as:

$$\sum_{x_{ij}} (u_i u_i^T + \lambda I). v_j = V^T.X_i^*$$

In Equation (2), $X_i^*$ represents $i^{th}$ column of $X$ and in Equation (3), $X_i^*$ represent $i^{th}$ row. The block matrix representation (4) of the data matrix is based on concatenation of blocks/sub-matrices of equal dimension as shown below:

$$X = \begin{bmatrix}
X_{11} & X_{12} & \cdots & X_{1j} & \cdots & X_{1J} \\
\vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\
X_{I1} & X_{I2} & \cdots & X_{IJ} \\
\vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\
X_{I1} & X_{I2} & \cdots & X_{IJ} \\
\vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\
X_{I1} & X_{I2} & \cdots & X_{IJ}
\end{bmatrix}$$

(4)
The Block Matrix Factorization (BMF) approach shown in Figure 1 considers each block as an individual matrix, factorizes for few iterations and takes the latent features of each of the blocks as a starting point for computation of latent features for relevant blocks afterwards. Figure 2 demonstrates simple example with 4 blocks each factorized individually and then combined together to form \( U, V \). After first sub-matrix \( X_{11} \) is approximated for one iteration, the resultant latent feature sub-matrices \( U_1 \) and \( V_1 \) are utilized during the MF of all relevant sub-matrices \( X_{ij} \) approximated subsequently; i.e., at every step resultant \( U_i \) and \( V_i \) are approximated from \( X_{ij} \), the latent feature matrix \( U_i \) is used further by all the sub-matrices \( X_{ij+1}, X_{ij+2}, \ldots, X_{iJ} \) lying on the same row as that of \( X_{ij} \).
Similarly, $V_j$ is used by all the sub-matrices $X_{i+1,j}, X_{i+2,j}, ..., X_{I,j}$ lying on the same column as that of $X_{ij}$. As the process repeats, the ALS convergence of each individual sub-matrix converges closer to regular MF. The optimization problem of BMF at an element level remains the same as Equation (1); while the update equations for for $i^{th}$ row of $U$ remains the same as Equation (2); and similarly the update equations for for $j^{th}$ column of $V$ remains the same as Equation (3).

The optimization problem of BMF at block level of $X_{ij}$ can be adapted from Equation 1 and be represented as:

$$\min_{U, V} \sum_{X_{ij}} (X_{ij} - U_i^T V_j)^2 + \lambda (\sum_{U} n_{U_{i,j}} ||U_i||^2 + \sum_{V} n_{V_{i,j}} ||V_j||^2)$$  

(5)

and the update equation for $i^{th}$ row of $U$ is given as:

$$\sum_{X_{ij}} (V_j V_j^T + \lambda I) U_i = U_i^T X_{i*}$$  

(6)

similarly the update equation for $j^{th}$ column of $V$ is given as:

$$\sum_{X_{ij}} (U_i U_i^T + \lambda I) V_j = V_j^T X_{*j}$$  

(7)

Equation 6 and 7 are adapted from 2 and 3 respectively. The approach can further be extended to sub-matrices which can further be divided into even smaller sub-matrices and process can be repeated till the sub-matrices are small enough to compute.

A broad outline of the approach is given in Algorithm 1. The $mf()$ function in the algorithm encapsulates implementation of regular MF at a block level for one iteration only. Considering $I$, $k$ as constants and as the two for-loops iterating STEPS number of times and iterating for each sub-matrix already contribute to $I$, $n$ and $m$ as discussed in Section I and hence the time complexity of the algorithm remains same as $O(n^4)$. Considering the block size as constant, the space complexity:

$$\approx (n \times m + n \times k + k \times m) \times c \rightarrow O(c)$$  

(constant)

**Algorithm 1 Block Matrix Factorization (BMF)**

**Require:** Input: Data matrix $X \in \mathbb{R}^{n \times m}$, number of features $k$

Initialize: latent feature matrices with random values $U \in \mathbb{R}^{n \times k}$, $V \in \mathbb{R}^{m \times k}$

Let $I, J$ be two constants such that $X$ is represented by $I \times J$ number of sub-matrices

Represent data matrix $X$ as block matrix with sub-matrices $X_{ij}$ where $i \in 1..I$ and $j \in 1..J$

Similarly, represent feature matrices $U, V$ as block matrix with sub-matrices $U_i, V_j$ where $i \in 1..I$ and $j \in 1..J$

Let STEPS be constant representing maximum iterations for MF and $\alpha$ be the learning rate, $\beta$ be the regularization factor and $\delta$ the minimum deviation of error between iterations

for step 1 to STEPS do
    for each block sub-matrix $X_{i\times j}$ do
        $U_i, V_j \leftarrow mf(X_{ij}, U_i, V_j, \alpha, \beta)$
        { $mf()$ function is a regular MF algorithm implementation, preferably implemented on a GPU to achieve ALS convergence }
    end for

    Terminate convergence if RMSE improvement is less than $\delta$
end for

return Latent feature matrices $U \in \mathbb{R}^{n \times k}, V \in \mathbb{R}^{m \times k}$
The implementation of the BMF algorithm is made available for public on Github at 1 along with code for all the experiments presented in section 3.

3 Analysis

Below experiments are conducted on shared hardware with Intel(R) Xeon(R) CPU E5-2640 v3 @ 2.60GHz. For the sake of comparison and evaluation, the experiments are conducted on 3 different block variations of a single MF implementation:

- 1 Block MF - considers entire data matrix as a single block
- 4 Blocks MF - splits the entire data matrix into 4 different sub-matrices
- 16 Blocks MF - splits the data matrix into 16 sub-matrices

The experiments in Section 3.1, 3.2 and 3.3 are conducted on a pseudo data set of $100 \times 100$ matrix with random values generated between 1 to 30. The $\alpha = 0.000001$, $\beta = 0.01$ are the learning, regularization parameters while $\delta = 0.01$ as the minimum difference of error between steps. The $k$ (number of latent features approximated) taken for the experiments is 10.

Table 1: Variable block level convergence

3.1 Root Mean Squared Error Analysis

Figure 3 provides comparative analysis of RMS error between different number of blocks used in MF. With the increase in overall number of iterations the ALS convergence of different block combinations converge nearly (within a delta) to same global optima. Although the converged RMSE values of different combinations (8.568 for 1 Block variant and 8.569 for 4, 16 Block variants) are not equal, the difference in convergence is within acceptable delta. Due to the need for transfer of latent feature learning between blocks, the convergence of multiple block variants of MF (between iterations 2-6 of Figure 3) are slightly slower than the single block variant.

3.2 Independent vs Dependent Blocks

The RMSE convergence shown in Section 3.1, Figure 3 is conducted on a randomly generated data matrix where the blocks are dependent on each other. The dependence of block can be arrived based on latent features derived at block level vs derived at entire matrix level. A dependent block’s latent features differ significantly from that of the relevant latent features of the entire matrix. However, as shown in example 2 each block could be independent from other blocks and could be a representative sample of the entire matrix. The RMSE convergence of such independent blocks follows a slightly different pattern as shown in Figure 4. The data set for independent blocks is generated with uniform value of 30 for a $100 \times 100$ matrix, where each block is independent and the latent factors are equal to the latent factors of the overall matrix. The initial RMSE values of multi-block variants of MF are slightly lower than the single block variant, as the RMSE value is observed to be reducing with each additional block factorized within the same iteration. Convergence of independent blocks can be expedited by increasing the number of iterations at block level for first block and the remaining blocks confirm the latent features resulted from first block. Where as increasing the number of iterations for dependent blocks result in sub-optimal convergence as described in Section 3.3.

3.3 Block level (approximate) convergence

Although BMF is optimal when each block is converged once and the respective $U_i$, $V_j$ values are passed on to converge following sub-matrices, an approximate convergence can be achieved with each blocked converged for more than one iteration for approximate but nearly optimal converge similar to SGD. The set of experiments under this section are conducted with data matrix split into 4 blocks with global optimum convergence RMSE of 8.568 Rows 1-4 of Table 1 demonstrate the deviation in RMSE convergence with different number of iterations of convergence at a block level. It is observed that the convergence moves away from global optima with additional iterations at block level, even though the overall number of iterations (block level iteration $\times$ steps) increase. Rows 5-8 of Table 1 demonstrate RMSE convergence with incrementally variable number of iterations of convergence at a block level. It is observed that the convergence moves closer to the optimum as we gradually increase the number of iterations of convergence at block
level. The results for Rows 5-8 shown in the table are obtained with \( n \) value < 8 to ensure the individual blocks are not fully converged. Choice of large \( n \) value leads to full convergence of the blocks with results similar to Row 9. Row 9 of Table 1 demonstrate RMSE convergence with each block approximated till convergence. It is observed that the convergence of entire matrix moves away from optimal value as the final block convergence has a significant impact on the overall outcome. Rows 10 of Table 1 demonstrate RMSE convergence with variable number of iterations, at a block level, decremented from high to 1. It is observed that the convergence matches the global optimum when the high value is lower than or equal to 8 for the given data set. As the high value is greater than 8, each block reaches full convergence causing challenges in transfer of latent features between blocks. The gradual decrease can be achieved by proportionally adjusting the number of block level iterations w.r.t to the ratio of RMSE improvement achieved in previous step.

3.4 Comparison against reference data sets

Table 2 lists the comparative analysis of time taken per iteration, number iterations to converge and Test RMSE for CPU implementation of BMF (1 Block, 4 Blocks, 16 Blocks variants of MF). All the experiments in Table 2 are conducted on standard hardware as mentioned at the beginning of the section 3. The \( k \) (dimension of latent feature vectors) value taken for all the experiments is 30 and \( \alpha = 0.000001, \beta = 0.01 \) are the learning, regularization parameters while \( \delta = 0.01 \) as the minimum difference of error between steps. The RMSE values of the experiments may not be comparable to that of the results from other papers, as the focus of the experiments are primarily to compare between block variants of MF. As observed, from the results, RMSE convergence of all Block variants are within delta. The slight increase in computational time with increase in number blocks can be explained by the need to frequent data transfer with the processing unit, which is a trade-off for achieving optimization in space complexity.
Figure 4: RMSE comparison of independent block combinations

| Dataset -> | MovieLense 100K | MovieLense 1M | Jester 4M |
|------------|----------------|--------------|-----------|
| n (users)  | 1000           | 27000        | 73496     |
| m (items)  | 1700           | 4000         | 100       |
| Training Time | 1 Block | 233.38   | 3029.63   | 943.84   |
|             | 4 Blocks       | 237.75       | 3213.45   | 1097.62  |
|             | 16 Blocks      | 281.06       | 3743.72   | 1281.62  |
| Test RMSE  | 1 Block        | 1.234        | 1.193     | 4.992    |
|             | 4 Blocks       | 1.234        | 1.186     | 4.997    |
|             | 16 Blocks      | 1.234        | 1.188     | 4.998    |

Table 2: Results of BMF against reference data sets
3.5 Parallelism

It is possible to approximate of $U, V$ matrices simultaneously by identifying the blocks that do not depend on each other. Figure 5 shows one such example with a $6 \times 6$ block matrix where blocks with same color code (and a number placed low right corner of the cell) that represent the sequence in which they are simultaneously converged; i.e each column of blocks is converged simultaneously and the blocks on the diagonal are processed first simultaneous step and then the convergence is moved on to blocks below them while considering the entire column as a loop starting from diagonal block. The approach is outlined in Algorithm 2 with block level parallelism with the use of multi-threaded MF implementation method $mf_{\text{parallel}}()$ which can be configured to run only one iteration for optimal result.

Parallel convergence could be much more effective if the number of iterations of convergence at a block level are more than one; i.e avoiding the need for frequent swapping of blocks to be processed by the processing unit. As detailed in Section 3.3, specifically Rows 7,8 and 10 of Table 1 it is feasible to identify variable iteration combination suitable for the given data set for, a nearly optimal, approximate convergence. Rows 1-2 of Table 3 demonstrate the advantage of parallel processing in terms of gain in computational time for reference data sets. Similarly, Rows 3-4 demonstrate RMSE convergence with variable block level iterations approach similar to the experiment in Row 10 of Table 1.

| Multi-threaded BMF Training Time (sec/iteration) | MovieLense 100K | MovieLense 1M | Jester 4M |
|-----------------------------------------------|----------------|--------------|-----------|
| 4 Blocks                                      | 136.22         | 1801.49      | 628.94    |
| 16 Blocks                                     | 128.26         | 1741.56      | 614.88    |

Test RMSE results for variable Block level iterations (8..1)

| 4 Blocks                                      | 1.234          | 1.186        | 4.998     |
| 16 Blocks                                     | 1.234          | 1.187        | 4.998     |

Table 3: Parallel, Variable BMF results against reference data sets

4 Conclusion

Proposed approach of BMF is a novel technique that provides computational advantage, with respect to memory needed, without compromising on RMSE convergence.
Algorithm 2 BMF with parallelism

\begin{algorithm}
\textbf{for} step 1 to STEPS \textbf{do}
\textbf{for} each column $j$ in block matrix $X$ \textbf{do}

\begin{align*}
l &\leftarrow \text{mod}((j + \text{step}), J) \\
U_l, V_l &\leftarrow mf_{\text{parallel}}(X_{il}, U_l, V_l, \alpha, \beta)
\end{align*}

{\textit{mf_{\text{parallel}}()} function is a thread based MF algorithm implemented to achieve ALS convergence. Caution need to be taken to ensure only one thread is executed per column by either fixing the thread pool to total number of columns and through synchronization}

\textbf{end for}

Terminate convergence if RMSE improvement is $\leq \delta$

\textbf{end for}
\end{algorithm}

The technique could be combined with NNMF [9], Maximum-Margin Matrix Factorization (MMMF) [10], Probabilistic Matrix Factorization (PMF) [11] and any other variants of MF just by changing the algorithm implemented within GPU. Although the experiments discussed largely focused on factorization of dense matrices, the approach has been extended to factorization of sparse matrices (as done in the case of reference data sets) just by modifying the underlying MF algorithm implementation to work on sparse matrices and similar experimental results were noted. The approach can also be extended to SGD convergence as implemented in [11] instead of ALS convergence so as to achieve further improvements in computational time by just modifying the underlying MF implementation. The approach can be applied to various domains (that leverage MF) ranging from recommender systems in collaborative filtering [12] to text mining [8]. Although we focused mostly on BMF with equivalent block sizes, further experimentation is possible with variable block sizes and variable CPU, GPU iteration combinations at different phases of convergence. Further research is also needed in identification of error limits for termination of convergence at block level.

References

[1] Wei Tan, Shiyu Chang, Liana L. Fong, Cheng Li, Zijun Wang, and Liangliang Cao. Matrix factorization on gpus with memory optimization and approximate computing. In ICPP, 2018.
[2] Lester W. Mackey, Michael I. Jordan, and Ameet Talwalkar. Divide-and-conquer matrix factorization. In J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, editors, Advances in Neural Information Processing Systems 24, pages 1134–1142. Curran Associates, Inc., 2011.
[3] Yongfeng Zhang, Min Zhang, Yiqun Liu, Shaoping Ma, and Shi Feng. Localized matrix factorization for recommendation based on matrix block diagonal forms. In Proceedings of the 22Nd International Conference on World Wide Web, WWW ’13, pages 1511–1520, New York, NY, USA, 2013. ACM.
[4] O. Axelsson. A general incomplete block-matrix factorization method. Linear Algebra and its Applications, 74:179 – 190, 1986.
[5] Robert L. Ellis, Israel Gohberg, and David Lay. Factorization of block matrices. Linear Algebra and its Applications, 69:71 – 93, 1985.
[6] W. Chen, Y. Li, B. Pan, and C. Xu. Block kernel non-negative matrix factorization and its application to face recognition. In 2016 International Joint Conference on Neural Networks (IJCNN), pages 3446–3452, July, 2016.
[7] Sven Koitka and Christoph M Friedrich. nmfgpu4r: Gpu-accelerated computation of the non-negative matrix factorization (nmf) using cuda capable hardware. The R Journal, 8(2):382–392, 2016.
[8] Volodymyr Kysenko, Karl Rupp, Oleksandr Marchenko, Siegfried Selberherr, and Anatoly Anisimov. Gpu-accelerated non-negative matrix factorization for text mining. In Gosses Bouma, Ashwin Ittoo, Elisabeth Mtais, and Hans Wortmann, editors, Natural Language Processing and Information Systems, pages 158–163, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.
[9] Daniel D. Lee and H. Sebastian Seung. Algorithms for non-negative matrix factorization. In Proceedings of the 13th International Conference on Neural Information Processing Systems, NIPS’00, pages 535–541, Cambridge, MA, USA, 2000. MIT Press.
[10] Nathan Srebro, Jason D. M. Rennie, and Tommi S. Jaakkola. Maximum-margin matrix factorization. In Proceedings of the 17th International Conference on Neural Information Processing Systems, NIPS’04, pages 1329–1336, Cambridge, MA, USA, 2004. MIT Press.

[11] Ruslan Salakhutdinov and Andriy Mnih. Probabilistic matrix factorization. In Proceedings of the 20th International Conference on Neural Information Processing Systems, NIPS’07, pages 1257–1264, USA, 2007. Curran Associates Inc.

[12] Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. Computer, 42(8):30–37, Aug, 2009.