On Factorizing Million Scale Non-Negative Matrices using Compressed Structures

Sudhindra Gopal Krishna, Aditya Narasimhan, Sridhar Radhakrishnan, and Chandra N. Sekharan

Presenter: Sudhindra Gopal Krishna

sudhi@ou.edu

School of Computer Science, The University of Oklahoma, Norman, OK

Presenter’s Bio

- Sudhindra Gopal Krishna is a final year Ph.D. Candidate in the School of Computer Science at the University of Oklahoma.
- His research foundation is based on democratizing resources to research via storing the data in a small footprint and performing required operations on the sorted data without having to extract them completely.
- Originally from Bengaluru, India, where he received Bachelor's Degree in Computer Science from Visvesvaraya Technological University, and a Master of Science in Computer Science from the University of Oklahoma, USA.
- Apart from his research and teaching at OU, he is engaged in outreach programs and have worked with K-12 teachers in the state of Oklahoma, to provide Computer Science education to High-School students under CodeSooner program, led by Dr. Sridhar Radhakrishnan.

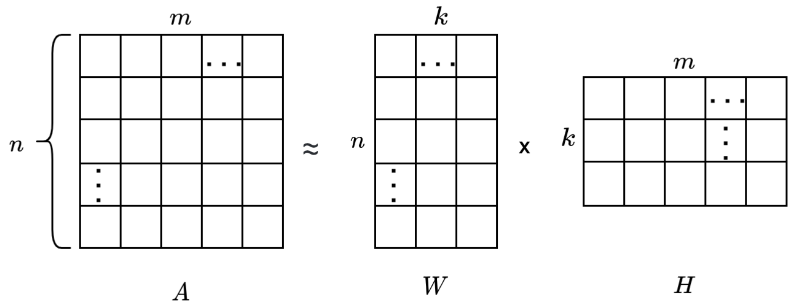
Contents

Introduction

- Matrix factorization is the process of decomposing a matrix into multiple matrices in order to simplify computations or extract meaningful information.
- Matrix factorization is a fundamental technique used in many areas of mathematics and computer science, including linear algebra, signal processing, and machine learning.
- Types: Some common types of matrix factorization include:
  - Singular Value Decomposition (SVD)
  - Principal Component Analysis (PCA)
  - Non-negative Matrix Factorization (NMF)
  - Latent Dirichlet Allocation (LDA)
- Applications: Some common applications of matrix factorization include image and video processing, collaborative filtering, and data compression.

Non-Negative Matrix Factorization

- Non-negative matrix factorization (NMF) is a type of matrix factorization where the matrices are constrained to contain only non-negative elements.
- NMF is often used as a tool for dimensionality reduction and feature extraction in machine learning applications, since it can produce interpretable and sparse representations of data.
- Some common applications of NMF include topic modeling, image and video processing, and text mining.



NMF Constraints

- W and H:
  - W is a matrix of size n x k, where n is the number of rows in V and k is the rank of the factorization.
  - H is a matrix of size k x m, where m is the number of columns in V and k is the rank of the factorization.
  - W and H are both non-negative matrices with all entries greater than or equal to zero.
- Frobenius norm:
  - The Frobenius norm of a matrix M is defined as the square root of the sum of the squared values of all the entries in M.
  - The Frobenius norm is commonly used as a measure of the distance between two matrices.

NMF Algorithms

- Some of the well-known sequential algorithms to solve the non-negative factorization are,
  - Multiplicative Update Algorithms
  - Gradient Descent Algorithms and
  - Alternating Least Squares Algorithms
- In this paper, we will evaluate the Multiplicative Update Algorithm defined by Lee & Seung

Multiplicative Update Algorithm

$$H \leftarrow \frac{H}{(W^T V)(W^T W H)}$$

$$W \leftarrow \frac{W}{(V H^T)(W H H^T)}$$

begin
    $W = rand(m, k)$
    $H = rand(k, n)$
    for $i : maxiter$ **do**
        $H \leftarrow H \;.\!*\; (W^T A) \;./\; (W^T W H + 10^{-9})$
        $W \leftarrow W \;.\!*\; (A H^T) \;./\; (W H H^T + 10^{-9})$

Figure 1. Multiplicative Update algorithm for NMF using the Frobenius norm as a cost function

NMF - Disadvantages of Lee and Seung's Approach

- Although the NMF approach proposed by Lee and Seung is widely used and has many benefits, there are also some disadvantages:
  - Local optima: The iterative procedure used in Lee and Seung's algorithm can sometimes converge to local optima rather than the global optimum.
  - Initialization: The performance of Lee and Seung's algorithm can be sensitive to the initial values of W and H.
  - Overfitting: If the rank of the factorization is chosen to be too high, NMF can overfit the data and capture noise rather than the underlying structure.
  - Interpretability: The basis matrices obtained from NMF can be difficult to interpret, particularly if the rank is chosen to be high.
  - Memory: The memory required to multiply two matrices requires tremendous amount of memory, as matrices are a 2-Dimensional data structure.

Solution

- In this paper, to solve the problem of memory requirement, we compress all matrices (A, W, & H).
- All matrix operations required to obtain final W & H, are all performed by partially deflating the data.
- To achieve this, in this paper we use Compressed Sparse Row (CSR), and Compressed Binary Trees (CBT), as storage mechanisms.

Background

- Paatero and Tapper (1994) proposed positive matrix factorization.
- Lee and Seung's NMF was inspired by Paatero and Tapper's work.
- Gonzalez and Zhang (2005) proposed an alteration to the multiplicative update algorithm.
- Lin (2007) proposed a modification that improved convergence.

Positive Matrix Factorization (PMF)

- Proposed by Paatero and Tapper in 1994.
- A matrix factorization method that restricts the factors to be non-negative.
- Inspired Lee and Seung's work on NMF.

Alternatives to Lee and Seung's NMF

- Gonzalez and Zhang (2005) proposed an alteration to the multiplicative update algorithm.
- Lin (2007) proposed a modification that improved convergence but at the cost of more operations per iteration.

Efficient Storage of Large Sparse Matrices

- The cost of storing zeros in large sparse matrices can be expensive and redundant.
- The sparsity of a matrix is defined as the ratio of the number of non-zero elements to the number of all possible elements.
- In this paper, we propose using our novel CBT algorithm and existing structures like CSR to efficiently store large sparse matrices.

Matrix Operations

- To obtain W and H, we need to perform several matrix operations such as,
    - Multiple Matrix Multiplication
    - Element-Wise Matrix Multiplication
    - Element-Wise Matrix Addition
    - Element-Wise Matrix Subtraction (Frobenius Norm)
    - Element-Wise Matrix Division
    - Matrix Transpose
- All operations should be performed on the compressed structure by the means of partial deflation

Element-Wise Matrix Operations

```
Input: Matrix A, Matrix B, Operation Op
Output: resultan_matrix C
if A.rowSize != B.rowSize or A.colSize !=
  B.colSize then
    Error: Matrix dimensions should be the
      same for both the matrices
for i in numberofRows do
    if A[i].rows == 0 and B[i].rows == 0 then
        C[i] = 0
        continue to the next row
    else if A[i] == 0 then
        C[i] = B[i]
        continue to the next row
    else if B[i] == 0 then
        C[i] = A[i]
        continue to the next row
    for aIndex in A[i] do
        for bIndex in B[i] do
            C[i][j] = A[i][j] "Op" B[i][j]
            Where "Op" = "+ or - or .* or ./"
return C
```

Matrix Transpose

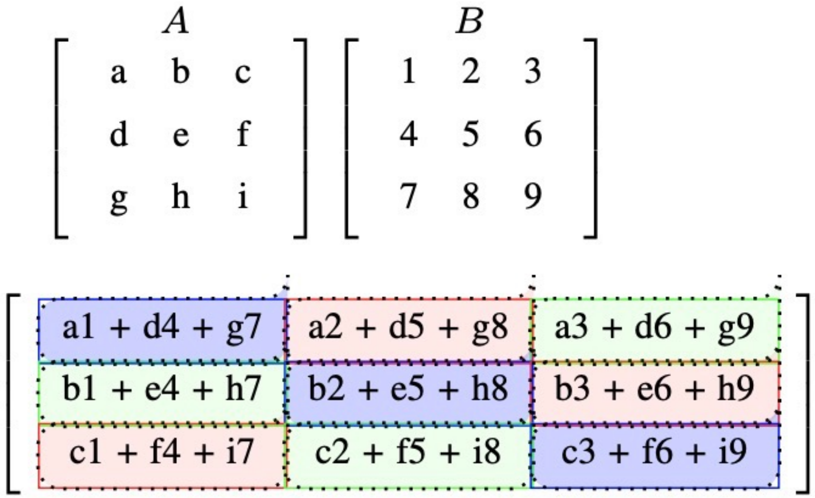


Fig. 1: Shows the working of $A^T \times B$, by storing the result in a pattern to eliminate the need of transposing the actual matrix.

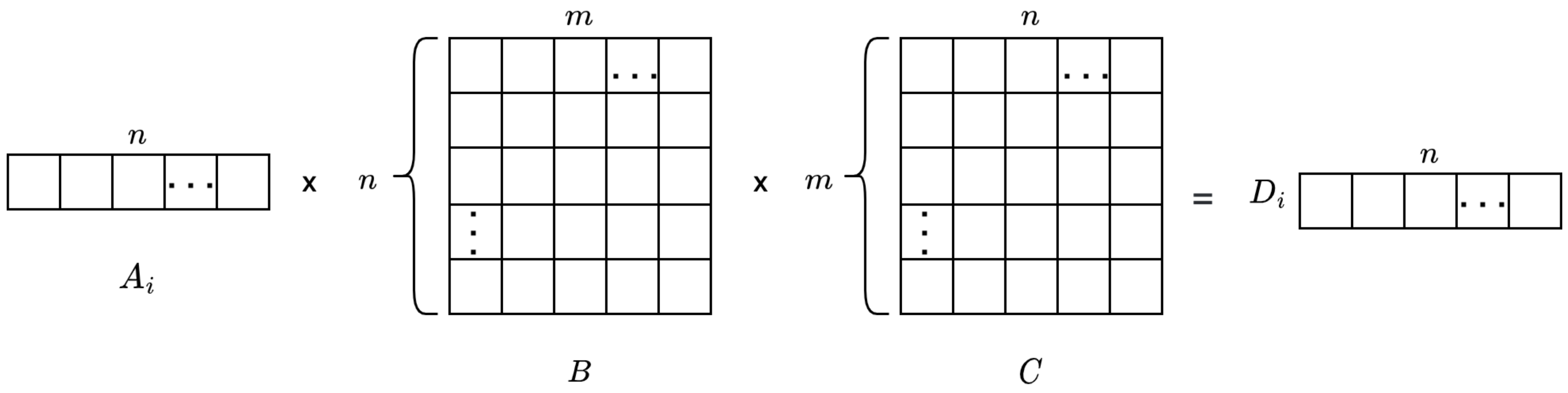
$$A \times B^T = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{cccc} 0 & 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 5 & 0 & 2 & 3 \\ 3 & 0 & 0 & 5 \\ 0 & 0 & 2 & 4 \\ 0 & 1 & 2 & 0 \end{bmatrix} \end{array} \times \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{cccc} 0 & 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 2 & 4 & 3 & 1 \\ 2 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 3 & 0 & 2 & 0 \end{bmatrix} \end{array} \quad (1)$$

$$\Rightarrow \begin{array}{l} r_0(A) \rightarrow \\ c_0(B) \rightarrow \end{array} \begin{pmatrix} 5 \\ \times \\ 2 \end{pmatrix} + \begin{pmatrix} 5 \\ \times \\ 4 \end{pmatrix} + \begin{pmatrix} 5 \\ \times \\ 3 \end{pmatrix} + \begin{pmatrix} 5 \\ \times \\ 1 \end{pmatrix}$$

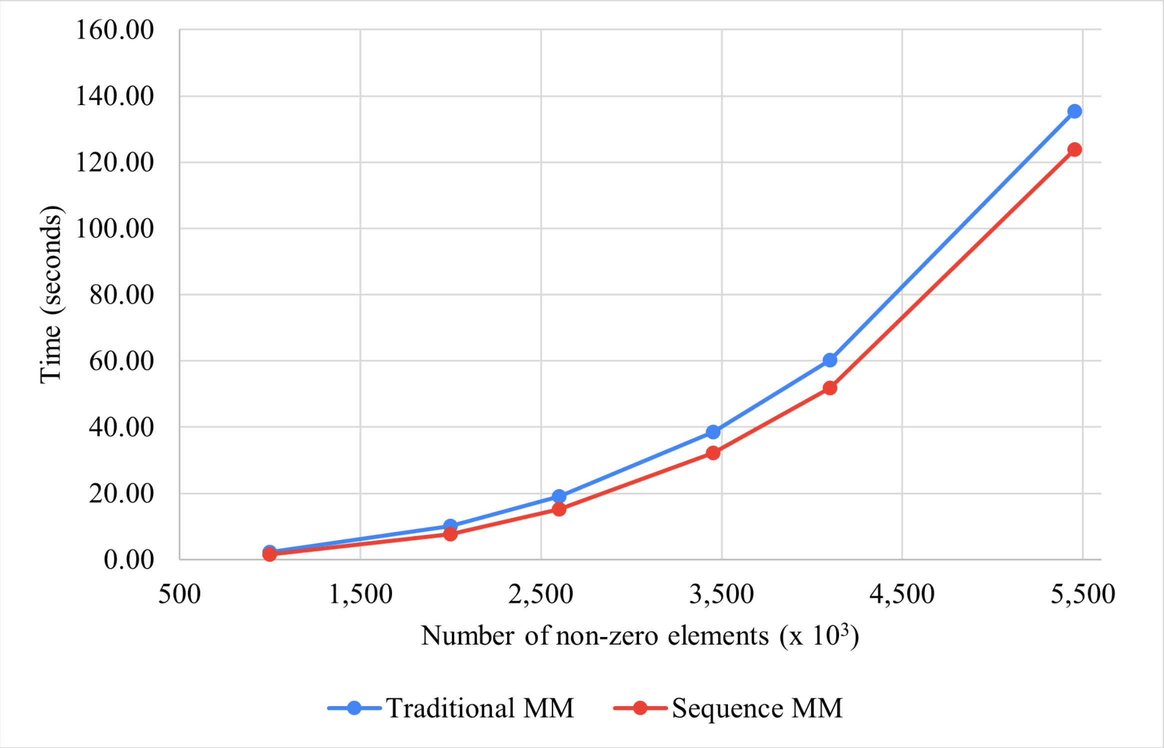
$$\Rightarrow c_0[C] = \{10\ 20\ 15\ 5\} \quad (2)$$

Equation 2, shows an example of $A \times B^T$, where the partial resultant of column $c_0[C]$, is obtained after multiplying the first row $r_0[A]$ of A, and virtually transposed the first column of B, in this case it is still $r_0[B]$.

Multiple Matrix Multiplication



Evaluating Multiple Matrix Multiplication



For a Million-By-Million Matrix with varying sparsity

Heuristics for Faster Convergence

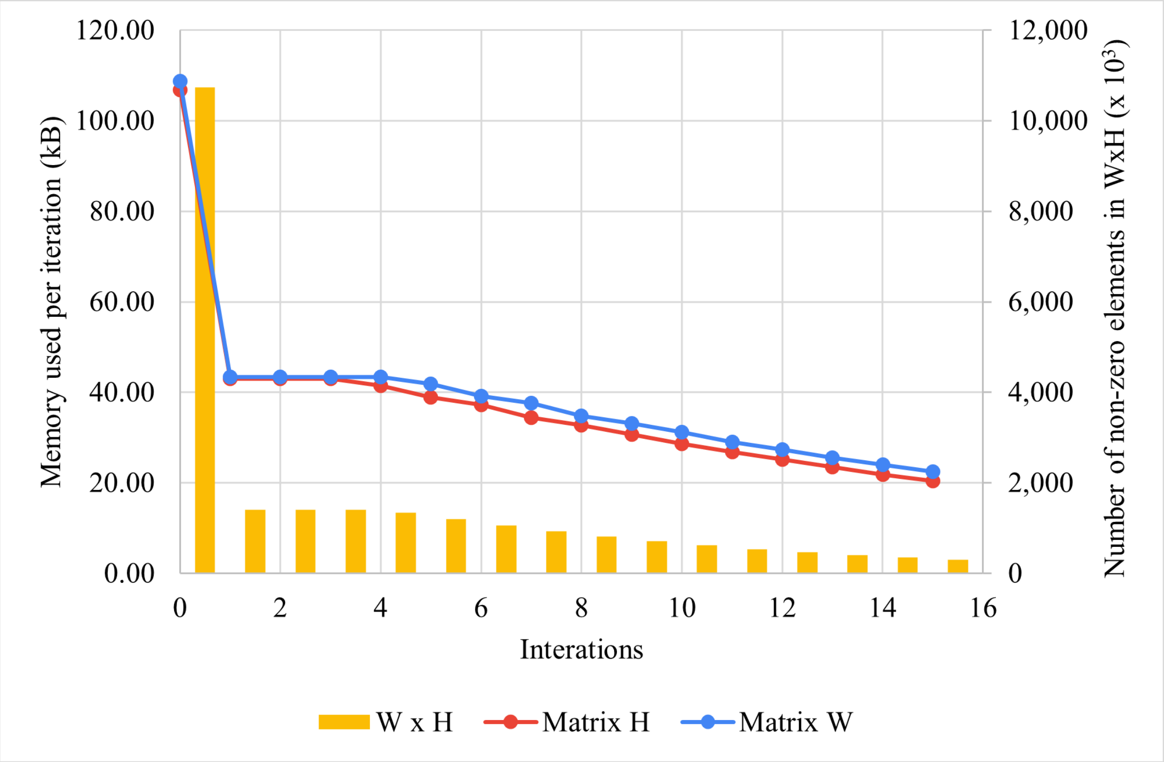
- One of the drawbacks of the multiplicative update approach is the convergence time and the iterations it takes to find an optimal solution.
- One way to make the algorithm faster is to reduce the number of non-zero values in the input matrix.
- A heuristic approach to reduce the number of non-zero values is to make specific values zero based on a threshold number of index positions per row.
- The decision to remove values at certain index positions will be based on two reasons: reducing the size of the compressed CBT structure and removing noise in the input data.
- This may lead to more loss, but the threshold will dictate the metric of the percentage of loss added to the already lossy factorization approach.
- The heuristic approach will not be optimal but will lead to reduced resource utilization.
- Space is reduced in the already compressed structure, and time to query the smaller CBT structure is reduced.

Evaluation

TABLE I: Shows the result of the factorization using CBT and CSR and the memory required to process the factors.

						W × H		Avg Mem/Iter		
Matrix A	**NNZ**	**Matrix Size**	**CBT**	**CSR**	**Inner Rank**	***NNZ***	***Matrix Size***	***Matrix***	***CBT***	***CSR***
2688×2688	23,089	55.12 MB	217.36 KB	216.23 KB	448	216.58 KB	216.51 KB	73.5 MB	0.54 KB	0.67 MB
5376×5376	57,752	220.5 MB	547.53 KB	546.68 KB	255	513.87 KB	526.46 KB	241.41 MB	0.29 KB	30 MB
21504×21504	1,385,198	3.44 GB	12.7 MB	12.98 MB	512	12.65 MB	12.95 MB	3.6 GB	13.1 MB	150 MB
43008×43008	998,531	13.78 GB	9.45 MB	9.53 MB	670	9.1 MB	9.98 MB	14.21 GB	9.92 MB	87 MB
65536×65536	1,460,048	32 GB	14.23 MB	14.05 MB	665	13.45 MB	14.12 MB	32.64 GB	14.80 MB	200 MB

Evolution of W & H



Conclusion and Future Work

- Million-scale matrix can be factorized directly on the compressed structure.
- Intermediate result can be eliminated using multiple matrix operations.
- Introduced element-wise matrix multiplication, division, subtraction, addition, and sequential multiple matrix multiplications.
- Traversing through the matrix in pattern can avoid an explicit transpose operation during matrix factorization.
- Heuristic relationship between inner rank and sparsity of factor matrices.
- Lower rank leads to smaller factors W and H.
- Future work: expand computation to ALS and GD, and Binary Matrix Factorization using compression algorithms.

Thank you

Questions?