Sampling and sketching methods for Matrix Factorization

Hitesh Kumar
hitesh.kumar@iitgn.ac.in
Computer Science Engineering,
IIT Gandhinagar, B.Tech 2022-26
India

Kishan Ved
kishan.ved@iitgn.ac.in
Computer Science Engineering,
IIT Gandhinagar, B.Tech 2022-26
India

Sumeet Sawale
sumeet.sawale@iitgn.ac.in
Computer Science Engineering,
IIT Gandhinagar, B.Tech 2022-26
India

ABSTRACT

In this report, we aim to explore non-negative matrix factorization performed on a sampled matrix using two approaches: Alternative Least Squares and Gradient Descent. Next, we move to a more specific problem, finding a sketch matrix B that approximates the matrix A^TA well ($B^TB \approx A^TA$).

1 INTRODUCTION

In Real-Life Applications, a Data Scientist usually deals with very huge matrices which cannot be stored on a local PC. Most common way to tackle this is to sample the huge matrix into some smaller component. However, after sampling, there is a loss of original data present in the matrix. There are many methods to reconstruct the matrix that is nearly equivalent to what it was before. This report focuses on the method named **Matrix Factorization** for this purpose. In matrix factorization we factorizes the matrix in two smaller components, which on product gives back the original matrix. But, here this has to be done on the sampled matrix with some technique to handle the missing values and then get back the original matrix. Also, use some metrics to calculate the quality of reconstruction. In this report, all of these operations were performed for an Image matrix.

2 MATRIX FACTORIZATION

Matrix factorization is a mathematical technique used to decompose a matrix into a product of multiple matrices, with the aim of capturing underlying patterns or structures within the original data. This process involves representing the original matrix as the multiplication of two or more matrices, typically of lower dimensions, which can provide insights into the relationships and features present in the data. For the sake of simplicity, let's consider the factorization into two matrices.
Consider a huge matrix A of size $m \times n$, where m and n are very large positive integers. We will decompose A into two smaller matrices W of size $m \times k$ and H of size $k \times n$ where $k <<< m$ and n, such that A can be approximated as:

$$A \approx WH$$

Fig. 1 shows the Image Matrix that was used for this project.

2.1 Matrix Sampling

Matrix sampling refers to the process of selecting a subset of elements from a matrix based on predetermined criteria or patterns. These criteria may encompass various methods such as random sampling, uniform sampling, or systematic sampling. The primary objective of matrix sampling is typically to decrease the size of the dataset while preserving its fundamental attributes, thereby facilitating more efficient analysis or processing without compromising the integrity of the data. The selection of random columns from the matrix also refers to matrix sampling (column sampling). The main focus will be on column sampling and the further computations will be performed on the column sampled matrix.

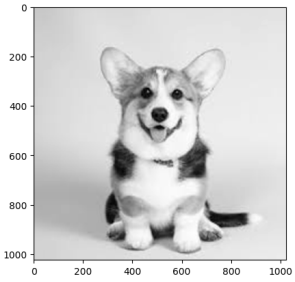

Figure 1: Original Image Matrix

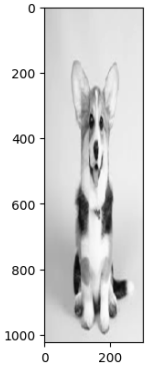

Figure 2: Sampled Image Matrix

The image matrix generated on performing column matrix sampling on the original image matrix is shown in fig. 2.

2.2 Non-Negative Matrix Factorization

Non-negative matrix factorization refers that the two factored matrices W and H must have non-negative entries. So, it is similar to matrix factorization, but with a constraint of factors having non-negative entries. This non-negativity makes the resulting matrices easier to interpret, and also it is used in many applications such as topic modeling, image processing, data compression, dimensionality reduction and speech recognition.

Now, the tricky part is that it has to be applied on the sampled matrix. That is discussed in the Implementation part of the report.

2.3 Kullback-Leibler (KL) Divergence

The Kullback-Leibler (KL) divergence, also known as relative entropy, is a measure of how one probability distribution diverges from a second, reference probability distribution. It's often used in statistics and information theory to quantify the difference between two probability distributions.

Specifically, for discrete probability distributions P and Q, the KL divergence from Q to P is calculated as:

$$D_{\mathrm{KL}}(P\|Q) = \sum_i P(i) \log\left(\frac{P(i)}{Q(i)}\right)$$

And for continuous distributions, the summation is replaced by integration:

$$D_{\mathrm{KL}}(P\|Q) = \int_{-\infty}^{\infty} p(x) \log\left(\frac{p(x)}{q(x)}\right) dx$$

KL divergence is asymmetric, meaning $D_{\mathrm{KL}}(P\|Q) \neq D_{\mathrm{KL}}(Q\|P)$. It is used in the first approach mentioning the use of Alternating Least Squares for finding the two factored matrices W and H.

3 APPROACHES

3.1 Approach 1: Using Alternating Least Squares

Alternating Least Squares (ALS) is an optimization technique used for matrix factorization, particularly in collaborative filtering tasks like recommendation systems. ALS iteratively decomposes a large matrix into two lower-rank matrices by minimizing the squared error between the observed and predicted values. The process alternates between updating one matrix while holding the other constant, and then vice versa.

Using KL Divergence as cost function for ALS we get the algorithm for ALS given below:

(1) Initialize W and H with random values.
(2) Update W and H using the following formulas:
(3) $W = W \times \frac{A \times H^T}{W \times H \times H^T}$
(4) $H = H \times \frac{W^T \times A}{W^T \times W \times H}$
(5) These updates minimize the Kullback-Leibler divergence between A and $W \times H$.
(6) Return the W and H matrices.

3.2 Approach 2: Using Gradient Descent

Starts with randomly initailizing the matrices W and H. The loss term is defined as the sum of the L2 norms of the difference between the columns of sampled A and the corresponding columns of the product $W \times H$ and in addition with the regularization terms for W and H. Using the Adam Optimizer for calculating the gradient and backpropagation for calculating the updated values of W and H. Also it is necessary to apply a hard constraint for entries of factored matrices to remain non-negative. After performing it for number of iterations, estimated W and H can be found on convergence of loss.

Since, given problem is not convex in nature therefore it is possible that the loss function might contain some local minimas and the algorithm might converge to local minima. Therefore a necessary update that is performed for escaping the local minima is boosting the learning rate for some time. The boosting of learning rate at local minima forces the function to escape the local minima and converge to more optimal solution.

If the loss does not change for a particular number of iterations (usually called as patience in Machine Learning), then we breaks from the loop of updation and return the found W and H.

3.3 Defining Accuracy

For observing and comparing the results the most important measure is finding accuracy. For this let's define accuracy. As for now, the project is based on image data, therefore we define accuracy as: 'The fraction of correctly predicted pixel values, where the correctly predicted pixel represents the pixel where the predicted value is not more than $'tol'$ far away from the actual value.' For all of the calculations the tolerance used for finding accuracy is 10.

3.4 Results

Here are the results obtained from both the methods, ALS and GD. In one method (ALS) results are not as expected but in other (GD) results that are observed are quite amazing that even after removing such a huge chunk of data, the original matrix is mostly recovered.

3.4.1 Alternating Least Squares Approach. Results are not too great for this method but for high k we can improve the reconstruction of the image. As shown in the fig. 3, overall accuracy increase with increase in value of 'k'. As mentioned accuracy is measured for the tolerance value of 10.

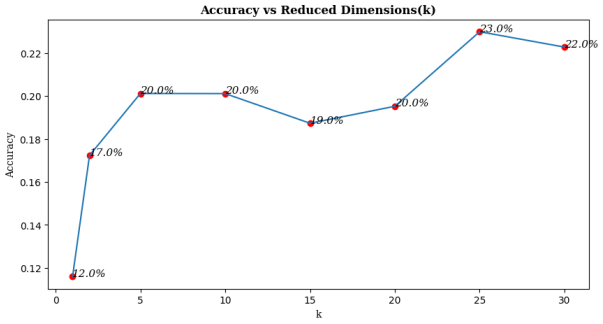

Figure 3: Accuracy VS 'k' from ALS

3.4.2 Gradient Descent Approach. This method worked very well as expected and the results shows the performance of the algorithm. For $k = 20$ and initial learning rate $lr = 0.1$, the change in loss vs iteration is shown in fig. 4. The peaks in the graph shows the presence of local minima in the loss function where the learning rate boosts and try to escape the local minima and reach more optimal solution. The resulting image reconstructed from the sampled matrix is shown in fig. 5 which is quite better w.r.t. the number of dimensions it is reduced to i.e. 20 from 300 (sampled image matrix 300×300) which are sampled from 1024 (original image matrix 1024×1024).

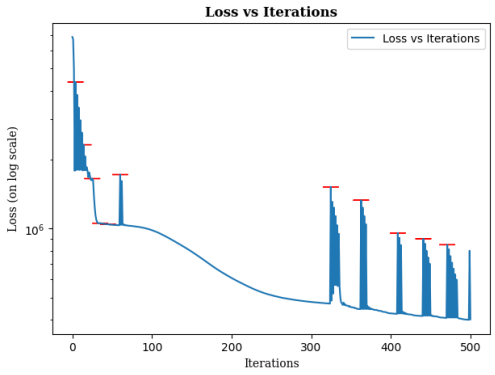

Figure 4: Loss VS Iterations for GD

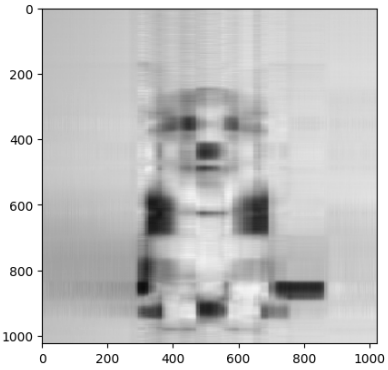

Figure 5: Reconst. Image Matrix from GD Approach for k = 25

On comparing the Accuracies for different values of Reduced Dimensions 'k', it is observed that accuracy increases with increase in 'k' as expected and most of the data is recovered from the two factored matrices W and H. Fig. 6 shows the same.

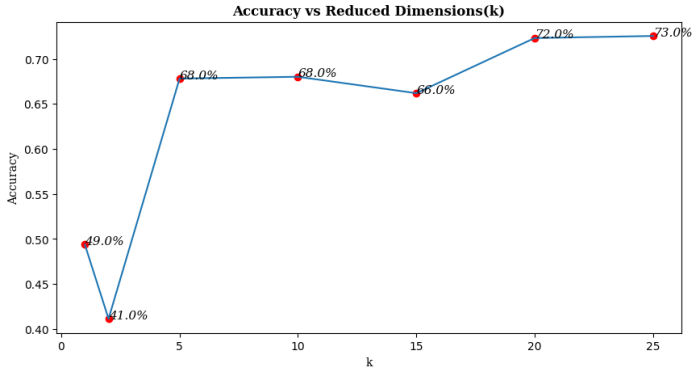

Figure 6: Accuracy Vs 'k' from GD

4 EXPLORING FREQUENT DIRECTIONS ALGORITHM ON A SAMPLED MATRIX

4.1 Background

In many real-life applications, the need arises to compute the product A^TA, where A is generally a very large matrix. The size of matrix A is often so substantial that computing A^TA becomes directly impractical due to computational constraints, such as the memory required to load the matrix A. A common scenario where this occurs is in recommendation systems, where A might represent a matrix with columns representing the ratings of different items and rows representing different users. Specifically, if A has dimensions $n \times d$, where n is the number of users and d is the number of items, then computing A^TA directly becomes computationally intensive and inefficient. In such cases, alternative strategies and algorithms are used to approximate or compute A^TA efficiently, allowing the analysis and processing of large-scale datasets in reasonable time frames. In this report, we will explore the performance of the Frequent Directions algorithm, which is given a matrix sampled from the original matrix A as input. The sampled matrix consumes less memory than the original matrix A. This matrix is sketched to generate a matrix B, such that $B^TB \approx A^TA$.

4.2 Objective

As described in the introduction, we are tasked with computing the matrix A^TA from a given matrix A. However, due to computational constraints such as memory limitations required to load the entirety of matrix A, directly computing A^TA becomes a challenging problem. Consequently, we resort to utilizing a sampled version of matrix A to mitigate these computational challenges.

The primary objective is to derive another matrix B, which, while significantly smaller in size compared to A, still offers a satisfactory approximation. This is accomplished through the application of the Frequent Directions algorithm, which operates on the sampled matrix as input. In essence, the goal is to discover a matrix B satisfying the relation:

$$A^TA \approx B^TB$$

Here, B is expected to possess dimensions $m \times d$, where $m \ll n$. This task revolves around the factorization of the matrix A^TA into two distinct matrices, B^T and B, aiming to ensure that the product of B^T and B serves as a credible approximation of A^TA.

4.3 Evaluation Metric and Algorithm

Frobenius Norm:

$$\|A^TA - B^TB\|_F$$

Frobenius norm of a matrix is the square-root of sum of squares of all entries in the matrix.

$$\|M\|_F = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}|M_{ij}|^2}$$

Algorithm 1: Frequent-directions

Input : $\ell, A \in \mathbb{R}^{n \times m}$
Output: B
$B \leftarrow$ all zeros matrix $\in \mathbb{R}^{\ell \times m}$;
for $i \in [n]$ **do**
  Insert A_i into a zero valued row of B;
  if *B has no zero valued rows* **then**
    $[U, \Sigma, V] \leftarrow \text{SVD}(B)$;
    $C \leftarrow \Sigma V$;
    $\delta \leftarrow \sigma^2_{\frac{\ell}{2}}$;
    $\Sigma^{\sim} \leftarrow \sqrt{\max(\Sigma^2 - I_\ell \delta, 0)}$;
    $B \leftarrow \Sigma^{\sim} V^T$;
  end
end

4.4 Sampling

Let's assume that accessing every row of matrix A is prohibitively expensive. In such cases, we can explore the following sampling techniques on matrix A:

- Uniform Row Sampling
  - Take rows 1,3,5,7… (difference of 2)
  - Take rows 1,4,7,11 … (difference of 3)
  - Take rows 1,5,9,13 … (difference of 4)
  - And so on … till we reach a difference of 10.
- Random Row Sampling
  - Randomly take 400 rows out of 500
  - Randomly take 450 rows out of 500
- Taking random permutations of rows

The difference between rows is referred to as "stride" in the context ahead.

4.5 Results: Uniform Sampling

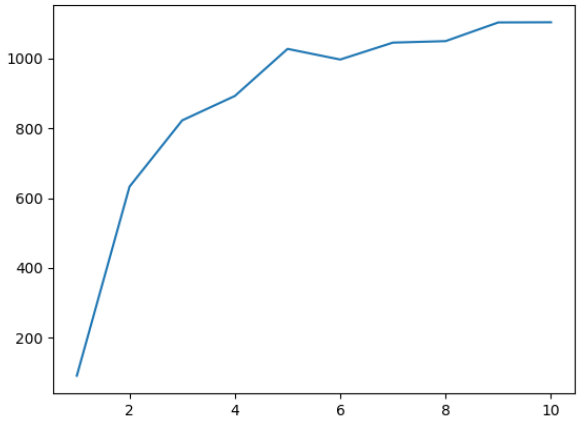

Figure 7: Plot of F-norm vs stride

4.6 Results : Random Row Sampling

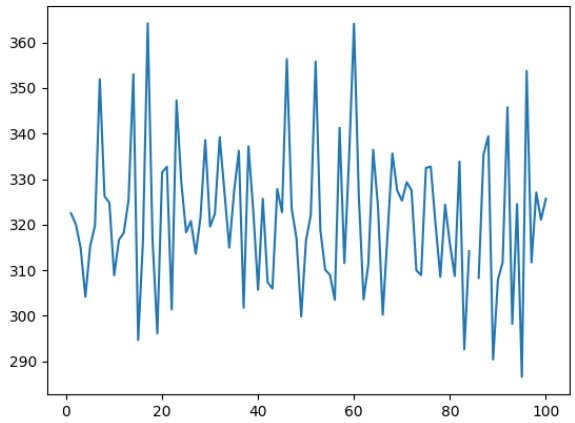

Figure 8: Random Row Sampling with 400 rows out of 500.

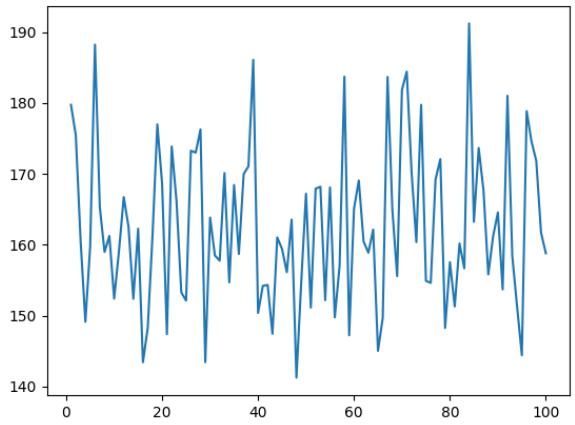

Figure 9: Random Row Sampling with 450 rows out of 500.

4.7 Results : Taking a random permutation of rows

4.8 Observations

- The F-norm (error metric) increases with the stride.
- The results show that uniform sampling does not work quite well. The main problem is that a lot of information is lost as we take very less number of rows into consideration. Let us shift to random row sampling, where we use more number of rows than we did in uniform sampling.
- The deviation from the mean is quite high for random sampling.

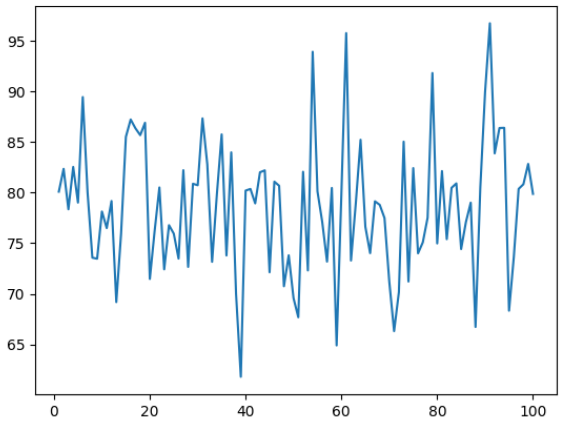

Figure 10: Rows in arbitrary order.

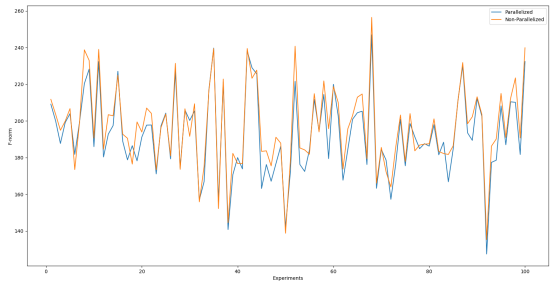

Figure 11: Performance on Parallelization

- Taking rows in any arbitrary order (ie; any random permutation) leads to a F-norm metric which is quite close to the that obtained by the actual algorithm.

4.9 Parallelization

Let us assume we have different machines for performing sketching using the Frequent Directions algorithm.
We divide matrix A such that the first $\frac{n}{2}$ rows go to machine 1 and the next $\frac{n}{2}$ rows go to machine 2. Additionally, we double the number of columns, $m \rightarrow 2m$. Consequently, each machine provides a $2m \times d$ matrix. We then combine both these matrices to form a $4m \times d$ matrix. This combined matrix is subsequently given to a machine for sketching, resulting in the creation of a $m \times n$ matrix B.

4.10 Results : Parallelization

See Figure 10

4.11 Observations

This parallelization technique helps us achieve performance similar to the actual algorithm.

Every experiment corresponds to a randomly generated matrix A to be sketched. In most experiments, parallelization demonstrates comparable performance to the non-parallelized algorithm. When A is too large to be processed by a single machine, instead of discarding rows (as in the case of sampling), we distribute parts of the matrix to two different machines, yielding similar results.
If we extend this parallelization to k machines, the approximate time complexity becomes $O\left(\frac{n \cdot l \cdot m}{k}\right)$, where l is the number of rows processed by each machine. Since the time complexity per row remains $O(ml)$, and in our case m becomes $2m$ and n becomes $\frac{n}{2}$, the overall time complexity remains unchanged.
Thus, parallelization does not increase the time complexity

5 ACKNOWLEDGEMENTS

We extend our sincere gratitude to Prof. Anirban Dasgupta for his invaluable advice and to TA Progyan Das for his dedicated assistance and insightful contributions.

REFERENCES

[1] Edo Liberty, Franco Woolfe, Per-Gunnar Martinsson, Vladimir Rokhlin, and Mark Tygert. [n. d.]. Simple and Deterministic Matrix Sketching. ([n. d.]). https://www.cs.yale.edu/homes/el327/papers/simpleMatrixSketching.pdf
[2] Wikipedia contributors. 2024. Kullback–Leibler divergence — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/wiki/Kullback%E2%80%93Leibler_divergence [Online; accessed 1-May-2024].
[3] Wikipedia contributors. 2024. Non-negative matrix factorization — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/wiki/Non-negative_matrix_factorization [Online; accessed 1-May-2024].