CS-238
Data Science Project

Sampling and Sketching Techniques for Matrix Factorization

Hitesh Kumar

Kishan Ved

Sumeet Sawale

Why Matrix Factorization?

- In Real World, we get expose with huge matrices which we can’t store in memory.
- Use Matrix Factorization to factorize the original matrix into two small components which we can store in memory and they give a better approximation of the original matrix

A huge matrix A of size m x n, $A \rightarrow m \times n$

Suppose 2 matrices W and H such that,

$$W \rightarrow m \times k \;\&\; H \rightarrow k \times n \;\mid\; k << m \;\&\; k << n$$

Let, $A' = WH$

Then, $A \approx A'$

Non-Negative Matrix Factorization

- Added a constraint that the factored matrices W and H must contain non-negative entries.
- Used in various fields such as:
  - Topic Modelling
  - Image Processing
  - Data Compression
  - Dimensionality Reduction
- Kullback-Leibler Divergence, is used to solve this problem.
- Let’s see what does Kullback-Leibler Divergence tells

Kullback-Leibler Divergence

- Measure of how one Probability Distribution diverges from a second, expected Probability Distribution
- It is always non-negative and zero only when two Distributions are identical
- Non symmetric, i.e. $KL(P, Q) \neq KL(Q, P)$
- Therefore it is used as cost function in Non-negative Matrix Factorization to measure the difference between the original matrix and its approximation through factorization
- Formula for KL divergence between two matrices:

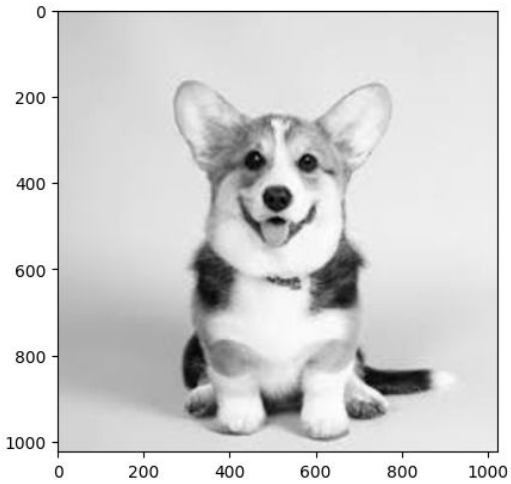
$$D_{KL}(A||A') = \sum_{i,j} (A_{ij} \log \frac{A_{ij}}{A'_{ij}} - A_{ij} + A'_{ij})$$

Sampling the Matrix

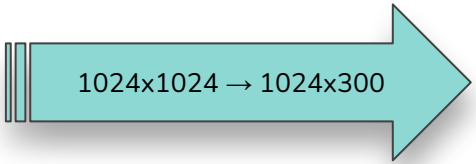
- Selecting the Subset of the Matrix. It could be the subset of columns, rows, etc.
- Used to reduce the data, because we can load the subset of the data in memory.
- Types:
  - Random Sampling
  - Stratified Sampling
  - Uniform Sampling
  - Cluster Sampling

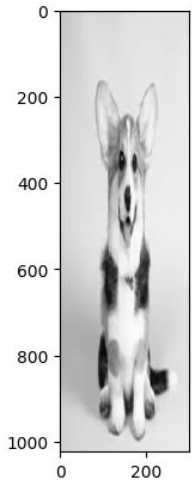
Random Sampling

- Randomly selecting ‘c’ number of columns from the true matrix and construct a Sampled Matrix out of it to load on the memory.
- Use the Sampled Matrix for further operations.



Original Image Matrix





Sampled Matrix

Alternating Least Square Approach

- Using Alternating Least Squares Algorithm with Kullback-Leibler Divergence as cost function for finding the two factorised matrices W & H.
- jth column of H can be written as a least square solution of W^TW and jth column of $W^TA_{sampled}$, if Asampled contains jth column of A after sampling.

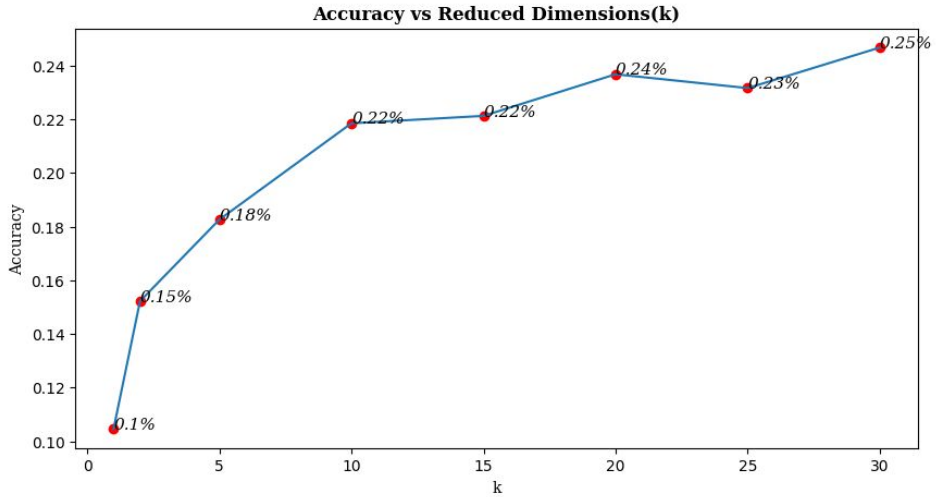
$$H[:,j] = LS\left(W^TW, W^TA_{sampled}[:,j] \right)$$

- ith row of W can be written as a least square solution of HH^T and ith row of $HA_{sampled}^T$, if Asampled contains ith row of A after sampling.

$$W[i,\ :] = LS\left(HH^T, HA_{sampled}{}^T[i,\ :] \right)$$

Results from ALS Algorithm

- If {abs(A - WH) < 10} → correctly approximated value (as we are taking an Image Matrix, we can handle the error of 10 at any pixel)
- Then Accuracy is defined as the fraction of correctly approximated values to the total number of values, i.e. mxn.



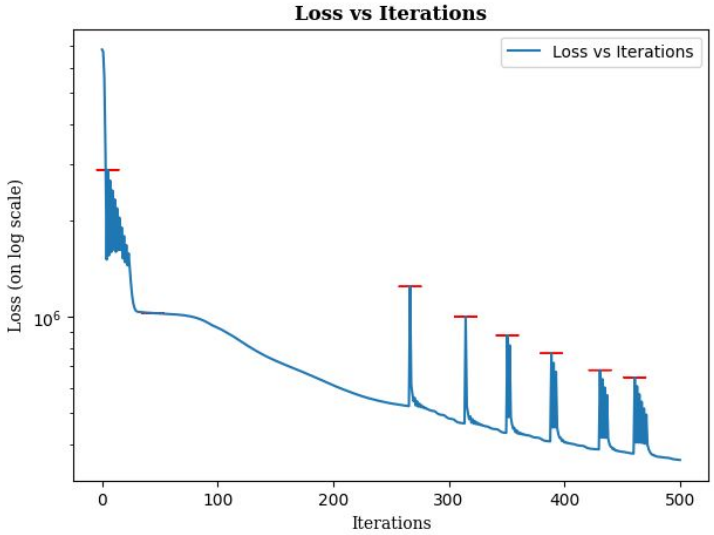
Gradient Descent Approach

- Hard Constraint → W & H will have non negative entries
- Loss → The sum of L2 norms of the difference between the jth column of Asampled and jth column of WH with the regularization terms for W and H.
- Optimizer → Adam Optimizer with learnable parameters → W & H
- Hyperparameters → Regularization Parameter, Learning Rate, Maximum number of iterations
- Returning W & H after scaling from 0 to 255 (pixel values)

Results and Observations

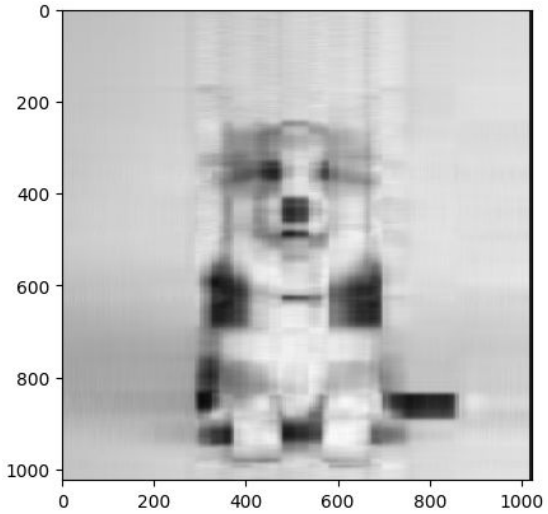
- For k = 20, (W → m x k, H → k x n) we get the plot of Loss Vs Iterations as

- Since, we have a Non-Convex Loss Function, the Peaks shown here in the graph are the local minima of the error function.
- Whenever the error function reaches a local minima, we tried to increase the learning rate so that it can get out of the local minima and can reach the global minima.
- This is the reason why the error is increasing at some points in the graph.



Results and Observations

- With the given conditions (k = 20, lr=0.1) we got the Accuracy of about 76.64%.
- The approximated Image matrix look like this



- As this is a Non Convex Problem we can’t say that this is the best solution, and with some more hyperparameter tuning and with different Initialization we can even get better results.

Goal: Finding A^TA

- In many real life applications, we need to find A^TA. However, A is generally a very large matrix. It is so large that A^TA cannot be computed quickly.
- An example of this is in recommendation systems, where A might be a matrix having columns representing ratings of different items, and rows representing different users.

$$A \rightarrow n \times d$$

n = Number of users

d = Number of items

Solution: Sketching

- The objective is the find another matrix B which is significantly smaller than A, but still approximates it well. In this context, we want to find a matrix B such that:

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

- The matrix B should be of shape m X d, where m << n.
- This problem is essentially factorizing the matrix A^TA into 2 matrices, B^T and B, such that the product of B^T and B is a good approximation of A^TA.

Algorithm: Frequent Directions

Input: ℓ, $A \in R^{n \times m}$

$B \leftarrow$ all zeros matrix $\in R^{\ell \times m}$

for $i \in [n]$ do

    Insert A[i] into a zero valued row of B

    if B has no zero valued rows then

        $[U, \sum, V] \leftarrow SVD(B)$

        $C \leftarrow \sum V^T$ # Only needed for proof notation

        $\delta \leftarrow \sigma^2_{\ell/2}$

        $\sum^{\vee} \leftarrow sqrt(max(\sum^2 - I_{\ell}\delta, 0))$

        $B \leftarrow \sum^{\vee} V^T$ # At least half the rows of B are all zero

    end if

end for

Return: B

Evaluation metric

- The Frobenius norm:

$$||A^T A - B^T B||$$

Credit - Paper from Yahoo! Labs

- The mentioned algorithm has been taken from the paper: Simple and Deterministic Matrix Sketching by Edo Liberty (Yahoo! Labs).

- As mentioned future work of this paper,

  “Another improvement might enable a Frequent-directions-like algorithm that can process the entries of the matrix in an arbitrary order and not only row by row. This is important for recommendation systems. In this setting, user actions correspond to single non-zeros in the matrix and are presented to the algorithm one by one in an arbitrary order. New concentration results show that sampling entries (correctly) yields good matrix sketches, but no deterministic streaming algorithm is known.”

The problem

- Let us assume that it is too expensive to go to every row of A. Can we use lesser rows?

  We explore the following sampling techniques on matrix A

  - Uniform row sampling
  - Random row sampling

- Will it make much difference if we feed all rows of A in any random permutation to the algorithm?

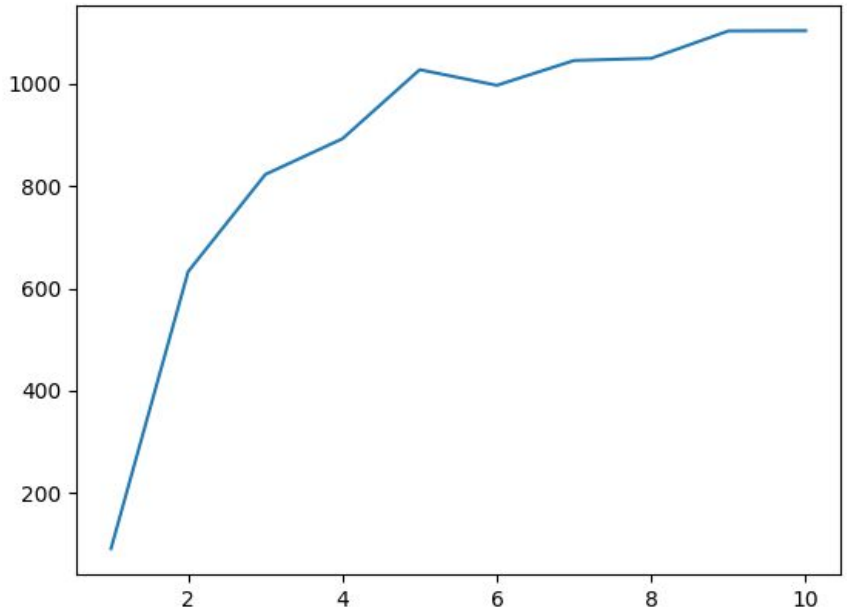
Uniform Row Sampling

- 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.

- Let us call this difference “stride”.

- Result: The F norm of $||A^TA - B^TB||$ increases with an increase in the difference between rows.

Plot of F-norm vs stride

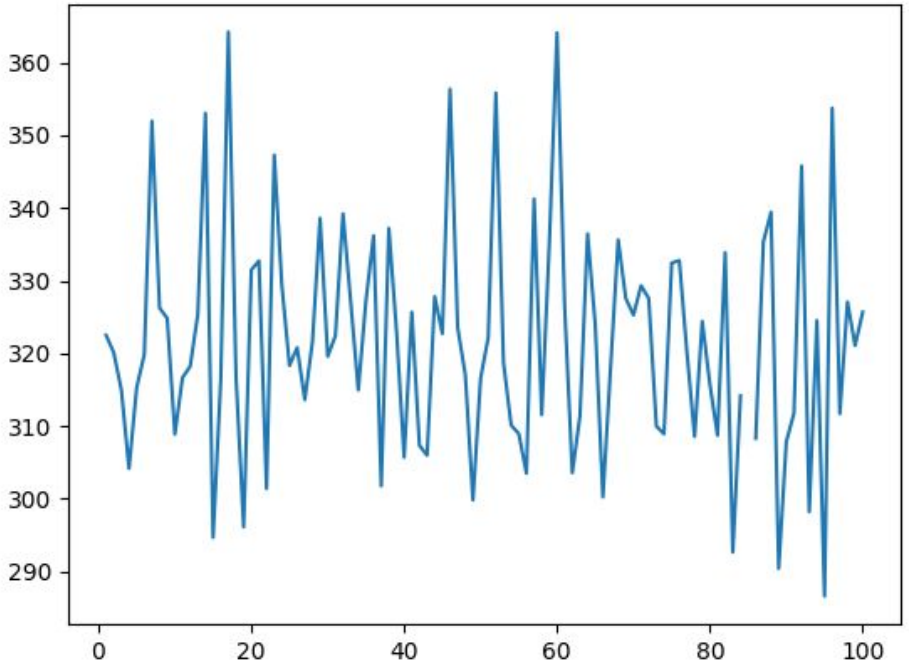


Random Row Sampling

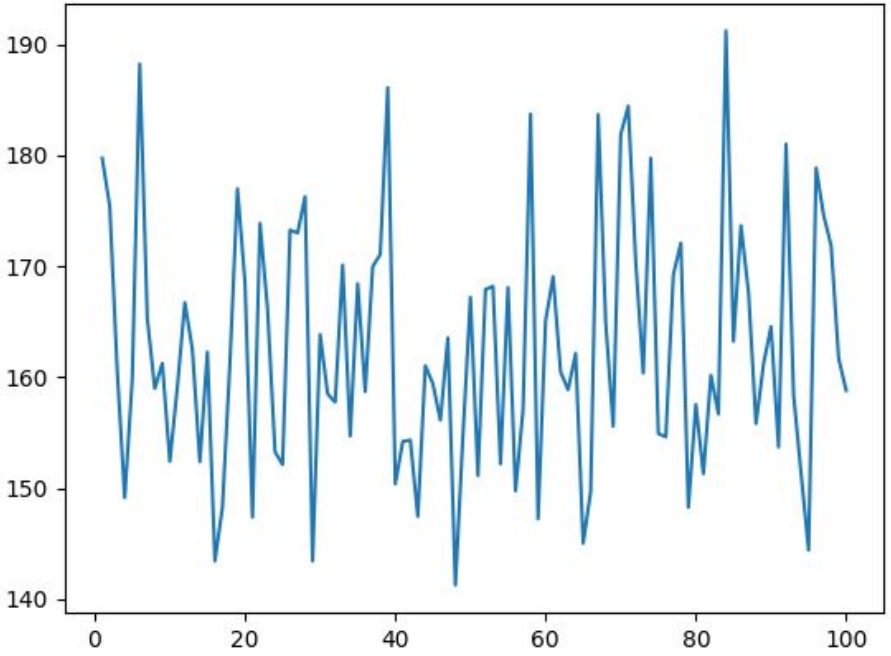
- This shows that uniform sampling does not work quite well. The main problem is that a lot of information is lost as we take very less number of rows into consideration.

- Let us shift to random row sampling, where we use more number of rows than we did in uniform sampling.
- Randomly take 400 rows out of 500
- Randomly take 450 rows out of 500

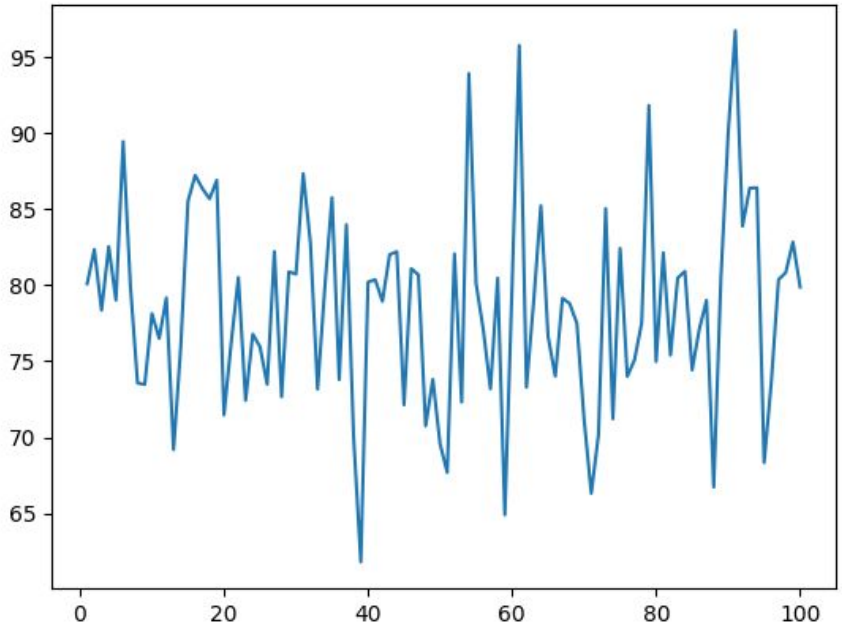
Random Row Sampling: 400 rows



Random Row Sampling: 450 rows



Rows in arbitrary order



Observations:

```
kishan@kishan-IdeaPad-3-15IIL05:~/Desktop/frequent-directions$ python exampleUsage.py
Frobenius norm between matrices AT.A and BT.B on performing simple sketching: 80.99795425739332
Average F norm on samling 400 rows 298.5378489177682
Average F norm on sampling 450 rows: 162.52713527344673
Average F norm on taking a random permutation of matrix A: 78.60286289468353
```

- The standard deviation is quite high for random sampling

- 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.

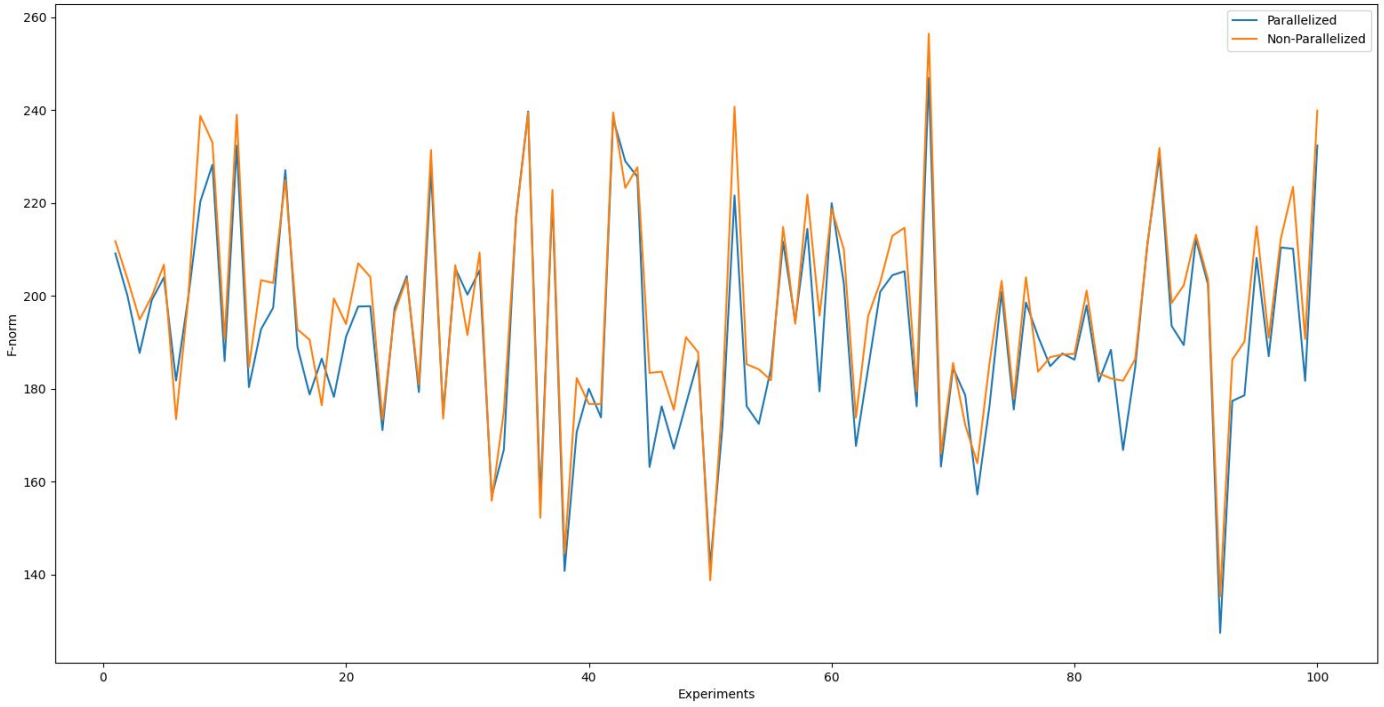
Idea: Can we parallelize this?

- Let us explore a technique to parallelize this process into 2 different machines.

- We divide A such that the first n/2 rows go to machine 1 and the next n/2 rows go to machine 2.

- We also change m to 2*m. This means that each machine gives us a 2*m X d matrix from each machine.
- Now, we combine both these matrices to form a 4*m X d matrix. This is now given to a machine, to sketch, thus, finally making a m X n matrix, which is the matrix B.

Surprising Results!



Observation

- Every experiment corresponds to a randomly generated matrix A, which we are going to sketch.
- In most of the experiments, we observe that parallelization performs at par with the actual non parallelized algorithm.
- If the matrix A is so large that it cannot be given to a single machine, then instead of discarding rows (as we did in the case of sampling), we can pass parts of the matrix into 2 different machines and still get the similar result.
- If we parallelize this to k machines, then the approximate time complexity is about O(nlm/k). (As it is O(ml) per row). In our case, m was 2*m and n was n/2, so, the time complexity remains the same.

Thank You!