École polytechnique fédérale de Lausanne

Project N°2 Report

Randomized Nyström

Authors:
Tara Fjellman
Amal Seddas

Professor:
Laura Grigori

Contents

1 Introduction

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

2 Randomized Nyström low rank approximation

The goal of the Randomized Nyström Algorithm is to represent a given matrix $A \in \mathbb{R}^{n\times n}$ by some lower rank $k < n$ approximation. The starting point of the algorithm is a rank-ℓ approximation of A, in the form

$$A_{Nyst} := (A\Omega)(\Omega^T A\Omega)^\dagger(\Omega^T A) \in \mathbb{R}^{n\times n},$$

where $(\Omega^T A\Omega)^\dagger$ is the pseudoinverse of $\Omega^T A\Omega$ and $\Omega \in \mathbb{R}^{n\times\ell}$ is a sketching matrix. We assume $k < \ell < n$. To further reduce the rank of A_{Nyst} to k, we have two choices: approximate only the core matrix $\Omega^T A\Omega$ or the whole A_{Nyst}. In this project, we focus on the second approach. Algorithm 1 shows how this can be achieved.

Algorithm 1 Randomized Nyström approximation using the Cholesky decomposition

1: Let $C = A\Omega$ and $B = \Omega^T C$.
2: Compute the Cholesky decomposition: $B = LL^T$.
3: Solve the linear system using back substitution: $Z = C(L^T)^{-1}$.
4: Perform QR decomposition: $Z = QR$.
5: Compute the singular value decomposition (SVD) of R and truncate:

$$R = U\Sigma V^T \approx \tilde{U}_k\tilde{\Sigma}_k\tilde{V}_k^T.$$

6: Set $\tilde{U}_k = Q\tilde{U}_k$ (or equivalently $\tilde{U}_k = Z\tilde{V}_k\tilde{\Sigma}_k^{-1}$) and return $\tilde{U}_k\tilde{\Sigma}_k^2\tilde{U}_k^T \approx A_{Nyst}$.

To show why the result of Algorithm 1 is in fact a rank-k approximation of A_{Nyst}, we write

$$\begin{aligned}\tilde{U}_k\tilde{\Sigma}_k^2\tilde{U}_k^T &= Q\tilde{U}_k\tilde{\Sigma}_k\tilde{\Sigma}_k^T\tilde{U}_k^TQ^T \\ &\approx QRR^TQ^T = ZZ^T = A\Omega(L^T)^{-1}(L^T)^{-1}\Omega^TA^T \\ &= A\Omega(LL^T)^{-1}\Omega^TA^T = A(\Omega^TA\Omega)^\dagger\Omega^TA^T.\end{aligned}$$

However, here we have the inverse of $\Omega^T A\Omega$ instead of its pseudoinverse. Unfortunately, Algorithm 1 fails in Step 2 if $\Omega^T A\Omega$ is numerically singular. In this case, as in [**?**], we replace L by a square root of B in SVD form. Specifically, we write $B = U_B\Sigma_B U_B^T$ (since B is SPSD), and set $L = U_B\sqrt{\Sigma_B}U_B^T$. By construction, we have

$$LL^T = U_B\sqrt{\Sigma_B}U_B^T(U_B\sqrt{\Sigma_B}U_B^T)^T = U_B\Sigma_B U_B^T = B.$$

We then replace $(L^T)^{-1}$ with $(L^T)^\dagger$. This can be simply calculated as

$$(L^T)^\dagger = (U_B\sqrt{\Sigma_B}U_B^T)^\dagger,$$

where $\sqrt{\Sigma_B}$ is a diagonal matrix whose entries are given by

$$(\sqrt{\Sigma_B})_{i,i} = \begin{cases} \frac{1}{(\sqrt{\Sigma_B})_{i,i}} & \text{if } (\sqrt{\Sigma_B})_{i,i} \neq 0, \\ 0 & \text{otherwise.} \end{cases}$$

3 Sketching and Sketching Matrices

In this section, we provide a detailed overview of sketching and the sketching matrices utilized in this project. The concept of sketching involves transforming a high-dimensional matrix $A \in \mathbb{R}^{n\times n}$ into a lower-dimensional representation $A\Omega$, where $\Omega \in \mathbb{R}^{n\times \ell}$ is a tall and skinny sketching matrix. The goal of sketching is to embed the high-dimensional data into a reduced-dimensional space while preserving the essential geometric properties of the data.

More formally, given any two vectors x and y in the high-dimensional space, their sketched counterparts $\hat{x}$ and $\hat{y}$ should approximately preserve their inner product:

$$|\langle \hat{x}, \hat{y}\rangle - \langle x, y\rangle| \leq \varepsilon \|x\|_2\|y\|_2 \tag{2}$$

where $\varepsilon > 0$ is a small approximation error. However, achieving this exact preservation for all x and y is generally infeasible due to the reduced dimensionality ℓ. Instead, Ω is typically regarded as a random matrix, ensuring that eq. (2) holds with high probability $1 - \delta$, where $\delta < 1$.

In this project, we employ two specific types of sketching matrices:

3.1 Gaussian sketching

The Gaussian sketching matrix is random projection method in which the entries of the sketching matrix Ω are drawn independently from a standard normal distribution. Mathematically, it is defined as:

$$\Omega_{ij} \sim \mathcal{N}(0, 1).$$

This method is such that for an input matrix $A \in \mathbb{R}^{n\times d}$ and for any vector $x \in \mathbb{R}^l$:

$$(1 - \varepsilon)\|Ax\|_2^2 \leq \|A\Omega x\|_2^2 \leq (1 + \varepsilon)\|Ax\|_2^2,$$

with high probability, where $\varepsilon > 0$ is a small approximation error. This property ensures that pairwise distances between points in the projected space are approximately preserved.

The Gaussian sketching matrix can also be viewed as a random transformation that approximately satisfies the following subspace embedding property for any subspace $T \subseteq \mathbb{R}^l$:

$$(1-\varepsilon)\|v\|_2^2 \leq \|\Omega v\|_2^2 \leq (1+\varepsilon)\|v\|_2^2, \quad \forall v \in T.$$

3.2 Subsampled Randomized Hadamard Transform (SRHT)

BSRHT is a version of the Subsampled Randomized Hadamard Transform (SRHT) specifically designed for distributed architectures. For n being a power of two, the SRHT can be defined as

$$\Omega^T = \sqrt{\frac{n}{\ell}} RHD,$$

where: $H \in \mathbb{R}^{n\times n}$ is the normalized Walsh-Hadamard matrix; $D \in \mathbb{R}^{n\times n}$ is a diagonal matrix with i.i.d. random variables $\sim$ Uniform(± 1); and $R \in \mathbb{R}^{\ell\times n}$ is a subset of ℓ randomly sampled rows from the $n \times n$ identity matrix.

Now, for P different processors, BSRHT can be constructed block-wise from the SRHT as:

$$\Omega^T = \begin{pmatrix} \Omega_1^T \\ \Omega_2^T \\ \vdots \\ \Omega_P^T \end{pmatrix} = \sqrt{\frac{n}{P\ell}} \begin{pmatrix} D_{L1} & \cdots & D_{LP} \end{pmatrix} \begin{pmatrix} RH & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & RH \end{pmatrix} \begin{pmatrix} D_{R1} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & D_{RP} \end{pmatrix}, \tag{2}$$

with: $H \in \mathbb{R}^{n/P\times \ell/P}$ being the normalized Walsh-Hadamard matrix; $D_{Li} \in \mathbb{R}^{n/P\times n/P}$, $D_{Ri} \in \mathbb{R}^{n/P\times n/P}$ being diagonal matrices with i.i.d. Rademacher entries ± 1; and $R \in \mathbb{R}^{\ell\times n/P}$ being a uniform sampling matrix, sampling along the rows.

This structure is particularly suited for distributed computations, as it allows for parallelism while maintaining the theoretical properties of the SRHT.

4 Stability analysis

- bullet points about comments to include

5 Parallelisation

We parallelise the algorithm by distributing among the processors the computation of $A\Omega, \Omega^T A\Omega$, and all other matrix operations with complexities at least proportional to n.

The parallelisation of QR factorisation was the topic of the first project. We therefore included the relevant functions for it in the code folder.

The parallelisation of matrix products was however implemented from scratch. It was done as described in algorithm 2, by distributing matrices smartly among the processors.

Algorithm 2 Computes the matrix product of two matrices D and E in parallel.

Require: q is an integer and the square root of the number of processors, D is an $m \times m$ matrix distributed such that processor i has $D_{i//q,i \bmod q}$ and E is a $m \times n$ matrix distributed such that processor i has $E_{i//q}$. FullProd is true if we also want to compute $E^T DE$.

Ensure: $F = DE$ (and $G = E^T F$ too if FullProd is true).

$F_{ij} \leftarrow P_{ij} E_j$

if FullProd **then**

$G_{ij} \leftarrow E_i^F P_{ij}$

end if

Row-wise Sum-reduce : $F_i \leftarrow \sum_j^q F_{ij}$

Column-wise Gather on rank 0 : $F \leftarrow [F_1^T, ..., F_q^T]^T$

if FullProd **then**

Sum-reduce : $G \leftarrow \sum_{i,j}^{q,q} G_{ij}$

end if

This version of the matrix product allows for a fluid computation of both the $A\Omega, \Omega^T A\Omega$ products involved in the sketching and general matrix products between two matrices D and E which appear in the computation of the rank-k approximation.

The pseudo-code for the parallelisation of the randomized Nyström algorithm is then described in algorithm 3.

Algorithm 3 Randomized Nyström algorithm. The syntax was adapted from this Overleaf example.

Require: A is an $n \times n$ symmetric positive semidefinite matrix, Ω is a sketching matrix of size $n \times l$, and k is the rank of the approximation
Ensure: $[A_{Nyst}]_k$, the rank-k randomized Nyström approximation of A.

$C \leftarrow A\Omega$
$B \leftarrow \Omega^T C$
$L, \text{Failed} \leftarrow \text{Cholesky}(B)$ ▷ Locally on rank 0
if Failed **then**
  $U, \Lambda \leftarrow \text{EigDecomp}(B)$ ▷ Locally on rank 0
  $B_k^+ \leftarrow U(:, 1:k)\Lambda(1:k, 1:k)^+ U(:, 1:k)^T$
  $Q, R \leftarrow \text{QR}(C)$ ▷ Using TSQR
  $\hat{U}_k \leftarrow QU(:, 1:k)$ ▷ In parallel
  $[A_{Nyst}]_k \leftarrow \hat{U}_k \Lambda(1:k, 1:k)^+ \hat{U}_k^T$ ▷ In parallel
else
  $Z \leftarrow CL^{-T}$ ▷ Computed by substitution : $(LZ^T)^T = C^T$
  $Q, R \leftarrow \text{QR}(Z)$ ▷ Using TSQR
  $U_k, \Sigma_k, V_k \leftarrow \text{TruncSVD}(R)$
  $\hat{U}_k \leftarrow QU(:, 1:k)$ ▷ In parallel
  $[A_{Nyst}]_k \leftarrow \hat{U}_k \Sigma^2(1:k, 1:k)\hat{U}_k^T$ ▷ In parallel
end if

6 Experimental procedure

The presented results were obtained by running our scripts on the Helvetios cluster, by averaging over different runs to make results more robust and interpretable.

In general, due to the BSRHT making use of the Hadamard transformation, n was made vary as powers of 2 only. In reality one can always zero-pad the data to ensure that the Hadamard transformation can be applied. This was however avoided for performance analysis since it would have introduced unnecessary variation in the measurements.

6.1 Stability analysis

....

6.2 Performance analysis

For these results to have some general relevance, we tried considering realistic values of k and l. More specifically, k was taken often around $n/20$, while l was taken logaritmically spaced.

When running in parallel, the number of processors was limited to values $P = 2^{2s}$ with $s \in \mathbb{N}$. This was due to perfect square constraint of the parallelised matrix multiplication and the BSHRT's use of the Hadamard transform. The values of P we choose were 1,4,16,64. This also meant having to take $l \leq n/64$ integer for the

TSQR algorithm to at least have one block-row per processor at the start of its run. We also recall that to see speed-up with parallelisation we need the matrix to be tall-skinny. To have a good speed-up for a bigger l, while keeping numerical stability for ill-conditioned matrices another algo should be used.

7 Algorithm Performance

7.1 Sequential Performance

To explore the sequential runtimes of the implemented algorithms, we decided to do two longitudinal studies : first varying l for a selected value of n, and then varying n for a selected l. The associated results are represented in 1.

The main feature common to both plots is that the k rank approximation part of the computation represents in most cases as small minority of runtime (as expected, since it should be of order $l^3 + nk^2$). This would of course change if one were to take big values of k (i.e. if the data was really information-dense). It's runtimes also do not really seem to depend on the sketching method. This makes sense given the computations are the same regardless of which sketching method was used to obtain the B and C matrices. One could probably change this if one were to fully take advantage of the structure of the matrices involved in the BSRHT method (as one could use integer-float operations instead of float-float ones).

Looking more closely at both figures, it is clear that different behaviours are observed as a function of l and n. We discuss each in the following paragraphs.

l variation As a function of l we observe extremely different behaviours from Gaussian and BSRHT sketchings. Indeed : the curve for BSRHT looks basically flat, while the Gaussian one increases steadily. This is expected given the complexities of the algorithms respectively are of order $n^2 \log_2 n$ and $nl^2 + ln^2$. This gives a great advantage in using BSRHT if we are embedding information-dense spaces as it scales much better.

Coming back to the k-rank approximation runtimes, we notice that the expected considerable increase as a function of l is observed, even if as mentioned it stays relatively small when compared to the sketching runtime.

n variation As a function of n both algorithms show significant increase. BSRHT shows faster runtime growth. For $n = 1024$ it seems to take twice the time, while for $n = 2048$ it seems to take three times as much. This is a perfect match due to extra $\log_2(n)$ term in its complexity. This gives a considerable advantage in using Gaussian sketching when handling information-sparse within high dimensional spaces.

As for the k-rank approximation runtimes, the linear increase as a function of n is hard to see on the plot due to the different scale.

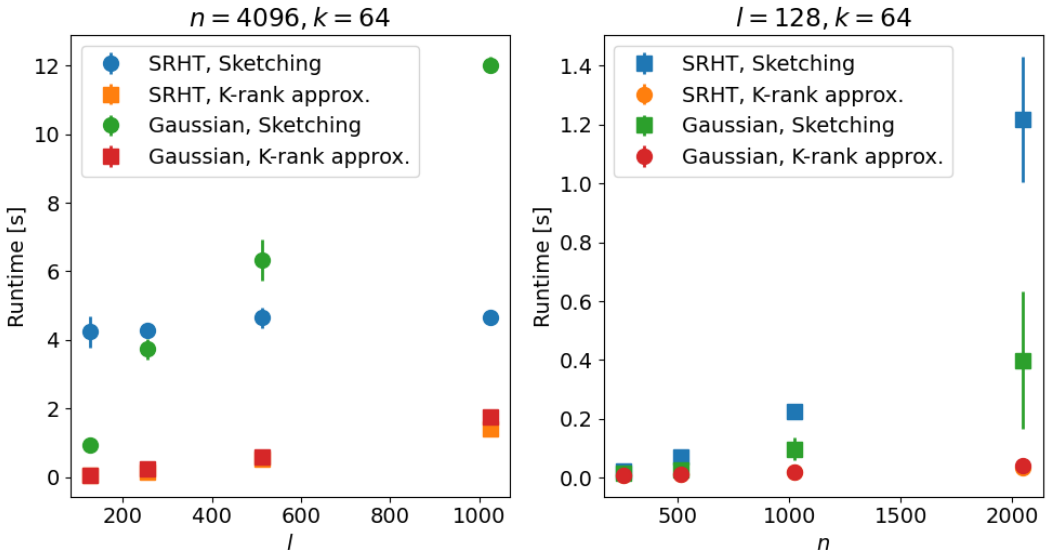

Figure 1: Runtimes associated to the tested sketching methods : as a function of l (left) and as a function of n (right). Runtimes are broken down in that associated to the computation of $A\Omega, \Omega^T A\Omega$ and that associated to the computation of the k rank approximation.

7.2 Parallel Performance

We now turn to the analysis of the parallel performance. The relevant plot for this section is 2. For the sake of structure, we analyse separately the small and the large runs.

Small run For the small run we first observe that some speed-up is displayed as a function of number of cores for both sketching methods. The speed-up is quasi-linear for BRSHT, while it saturates and back-fires for Gaussian sketching, due to the communication overhead. Given that the k-rank approximation part of the computation is smaller, it is most affected by the cost of communication, which actually dominates all throughout the core-number sweep. These behaviours are expected given that : (a) the computation is rather small to be run in parallel; (b) BRSHT is slower than Gaussian sketching for small values of l.

For the specific parameter values selected, Gaussian sketching is a factor ≈ 10 faster than BRSHT (except $P = 64$).

Large run For the large run the results are quite different. Communication is not an issue for the sketching part, meaning that quasi-linear speed-ups are observed for both methods. This means the parallelization is successful and scales well for big computations. We also observe initial speed-up for the k-rank approximation, though it is short-lived due to the computation still being quite small and due to that the matrix is not that tall-skinny (reducing the benefit in the QR decomposition).

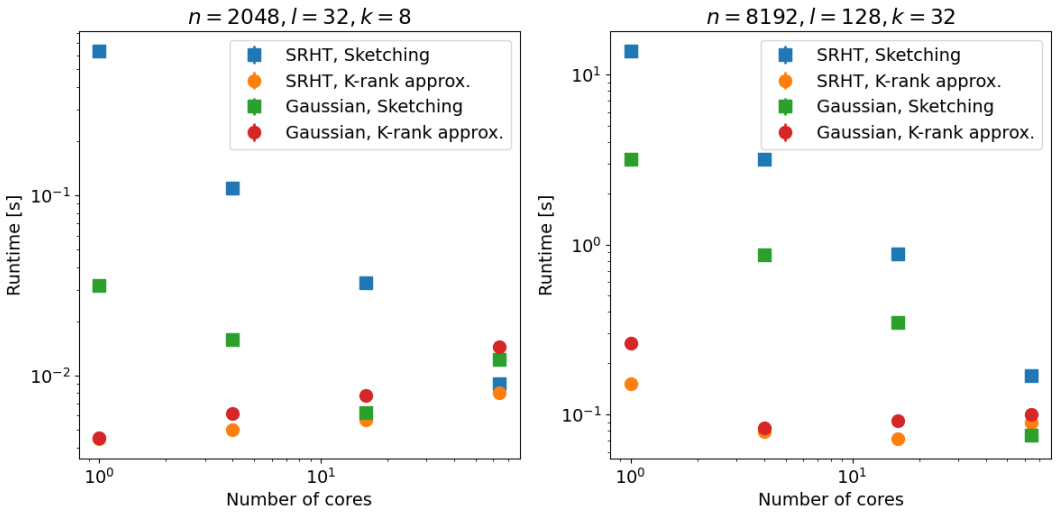

Figure 2: Runtimes associated to the tested sketching methods as a function of the number of cores : for a "small" example (left) and a "large" one (right). Runtimes are broken down in that associated to the computation of $A\Omega, \Omega^T A\Omega$ and that associated to the computation of the k rank approximation.

8 Conclusion

Aknowledgements

References