Compressed Climate Data on TileDB

Osama Dabbousi
osama.dabbousi@kaust.edu.sa

Abstract

This paper presents a novel approach to compressing the Red Sea re-analysis dataset, consisting of high-resolution GAN-generated temperature maps, using low-rank approximation via Singular Value Decomposition (SVD). We introduce techniques such as data zero-centering and quantization to enhance compression efficiency while maintaining a predefined error threshold. The dataset, stored as a combination of sparse and dense arrays in TileDB, is processed in manageable tiles to accommodate memory constraints. Through extensive experiments and ablation studies, we evaluate the impact of SVD rank, tiling dimensions, and error quantization on the compression ratio. Our results demonstrate significant improvements in compression when leveraging zero-centered data and quantization for the error array. Limitations and potential areas for future exploration are also discussed.

Keywords

SVD, Compression, Quantization

ACM Reference Format:
Osama Dabbousi. 2024. Compressed Climate Data on TileDB . In . , 3 pages.

1 Introduction

There are many instances where one may need to compress a large set of data. For example, to reduce memory requirements or reduce bandwidth strain during network transmission. Especially with the use of large sets of image or video data in research capacities for machine learning (ML) applications, there is a strong incentive to efficiently store data. In this paper, we explore the use of low-rank approximation using singular value decomposition (SVD) to reduce the size of the red sea re-analysis dataset. We explore ideas including the use of quantization for compression, the centering of data to improve SVD quality, and reason about ideal dimensions for SVD compression. We run ablation studies to verify the effectiveness of these methods, and present our final results. Lastly, we discuss weaknesses in our current implementation and future work that could build on our findings.

2 Methodology

2.1 Problem Formulation

For the purposes of this project, a subset of the red sea re-analysis [1] dataset was used- which is a set of GAN-generated images plotting temperature for a region surrounding the red sea. In particular the dataset was 4000 images at a resolution of 855x1215. Our goal for the project was to decompose this original dataset D into four sub-arrays. Firstly, using truncated SVD we decompose the data into three arrays U, S, and V, each of rank k « rank(D), such that $USV^T \approx D$. From there, we create an array E such that if we take $USV^T = D'$ then:

$$\forall i \quad \frac{|d_i - (d'_i + e_i)|}{range(D)} \leq \epsilon$$

for a given acceptable error value ϵ, which was chosen to be 10^{-2} for the purpose of this project. Lastly, we formulate the success of a given compression through its compression ratio ρ, calculated as:

$$\rho = \frac{\text{sizeof}(D)}{\text{sizeof}(U + S + V + E)}$$

2.2 Data handling and SVD computation

Given the size of the original data and its corresponding compressions it would prove infeasible to manipulate the data in RAM. As a result, the data was processed using TileDB [2] to allow for fast reading and writing to the disk. The data was stored in the (Time x Height x Width) dimensionality, with a float32 format, and tiles were defined with dimensions (200, 855, 1), representing a single column slice across 200 images.

The nature of TileDB sparse arrays is that data within a tile is stored contiguously and thus requires significantly less time to access than data across many tiles. As a result, the choice of tile dimension was made as a compromise between two requirements. The first was the method by which data was written to the array, and the second was the way it was read for use in compression.

While writing the original data to the array, images came in sequentially. As a result, reading in 200 images at a time meant loading the data across 1215 of the (200, 855, 1) tiles over a span of $4000/200 = 20$ iterations, which proved to be a manageable balance of tile-size and number of I/Os, and reduced redundancy during I/Os due to mismatched input/output size due to tile granularity. On the other hand, it was found during experimentation that SVD was most effective when applied on a number of pixels along the time-axis (i.e. a (4000, 855, 1) slice. As a result, the chosen (200, 855, 1) tile allows for slices to be taken along the time and height dimensions quickly. The choice of tiling along the columns versus the rows was made entirely arbitrarily, and we invite future works to further explore the impact of this choice.

From there, data was stored in the SVD arrays (U,S,V) according to the SVD dimensionality. For example, when the SVD was taken in batches of (4000x855x1), there were a resulting 1215 sets of (u, s, v) sub-arrays created. Each of them was stored sequentially in accordance with the order they were generated, resulting in three SVD arrays with dimension

$$U = (\#\text{ of SVD iterations, array height, k})$$

$$S = (\#\text{ of SVD iterations, k, 1})$$

CS245, December 2024, Thuwal, Saudi Arabia

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM

$$V = (\#\text{ of SVD iterations, k, array length})$$

where k is the chosen rank of the truncated SVD. Lastly, E was created as a TileDB sparse array with dimensions identical to D. E was chosen to be sparse because ideally the proportion of non-zero values in E would be low. We note that if E were dense, or if the sparse E was fully populated then a compression ratio > 1 would be impossible.

2.3 Further Improvements

In addition to experiments done to find ideal parameter values for k and the dimensionality of the SVD, three modifications to the original algorithm were tested.

The first modification was the use of quantization on the error array. By finding the minimum and maximum errors incurred during compression, we can use this information to create a linear map- taking the range [min_error-max_error] and mapping it into the range [0-255] as follows:

$$\hat{e}_i = \frac{e_i - \text{min\_error}}{\text{max\_error} - \text{min\_error}} * 255$$

which allows us to then represent the data using an unsigned 8-bit integer- reducing its memory footprint as a 32-bit float down by a factor of 4. Doing so naturally incurs additional truncation error as the 8-bit integer is significantly less expressive. However, we can quantify the maximum possible truncation error caused by this operation. The step size s of the quantized error can be calculated as

$$s = \frac{\text{max\_error} - \text{min\_error}}{255}.$$

The greatest error that can be incurred by quantization is half the step-size, in the case that the true value is between two steps. We also know that by definition $d_i = d'_i + e_i$ up to machine precision which for our context is negligible. Thus, with the introduction of the quantization error ϵ_{quant} such that $\hat{e}_i = e_i + \epsilon_{quant}$, we get

$$\begin{aligned}
\frac{|d_i - (d'_i + e_i + \epsilon_{quant})|}{\text{range}(D)} &\leq \epsilon \quad && \text{Substitute } \hat{e}_i = e_i + \epsilon_{quant} \\
\frac{|0 + \epsilon_{quant}|}{\text{range}(D)} &\leq \epsilon \quad && \text{since } d_i = d'_i + e_i \\
\frac{s}{2 \cdot \text{range}(D)} &\leq \epsilon \quad && \text{Substitute } \epsilon_{quant} = \frac{s}{2}
\end{aligned}$$

Given we can explicitly calculate s, range(D) and are given ϵ, we can verify if the quantization is guaranteed to be a safe operation, and apply it dynamically when it is. While in the typical case, quantization was reasonable for the error array E, when the use of quantization was tested on the arrays U, S, and V, the bound for error was found to be less well-behaved, and it was much more time-consuming to calculate the quantized form and verify its safety. Moreover, due to the additional incurred error in their truncation, we saw a nearly proportional increase in the size of E which rendered any additional compression by their quantization redundant. As a result, our final implementation forgoes the quantization of these terms.

The second modification was incredibly simple but had a drastic impact on performance. This modification was the zero-centering of the input image-data before SVD was calculated. It significantly improved the expressive-ness of the SVD at equivalent rank-k approximations. While the reason for this can't be explicitly proven, it is know that zero-centered data has more well-behaved principle components, and helps improve numerical stability in the SVD algorithm [3].

Similarly, experiments were done to test normalizing the standard deviation of the input data, however this appeared to have an adverse effect on compressibility. We believe this is due to the nature of our error calculation. Since error was defined relative to the total range of the data (range(D)), it incentivizes the compression algorithm to ignore values with smaller absolute variance. On the other-hand, normalizing the deviation of the data incentivizes SVD to treat variances in all dimensions equally.

Thus, the final implementation took advantage of the above two methods, excluding standard deviation calculations.

2.4 Results

Table 1 shows the experimental results for the compression before the implementation of zero-centering and quantization. The values for length/height/width represent the size of the SVD tiling as a function of the length of that dimension, while k represent the rank-k approximation size.

Table 1: Experimental results before quantization and zero-centering.

Exp #	Length	Height	Width	k	Compression Ratio
1	1/4	1/5	1/5	50	5.26
2	1/4	1/5	1/5	20	4.19
3	1	1	1/15	50	4.65
4	1/2	1/3	1/3	50	3.92
5	1/2	1/3	1/3	200	4.86
6	1/4	1/5	1/5	45	5.13
7	1/4	1/5	1/5	65	5.51
8	1/4	1/5	1/5	75	5.61

Table 2: Experimental results after quantization and zero-centering.

Exp #	Length	Height	Width	k	Compression Ratio
9	1/4	1/5	1/5	100	6.23
10	1/4	1/5	1/5	75	6.3
11	1/4	1/5	1/5	25	4.9
12	1/4	1/1	1/1215	50	7.6
13	1/4	1/1	1/1215	40	8.5
14	1/4	1/1	1/1215	30	9.2
15	1/4	1/1	1/1215	20	9.3
16	1/2	1/1	1/1215	40	10.44
17	1	1	1/1215	80	8.25
18	1	1	1/1215	40	11.8
19	1	1	1/1215	20	12
19	1	1	3/1215	80	15
20	1	1/855	1	35	11.8
21	1	1	5/1215	80	17.8

Table 2 represents the experimental results after these methods were implemented. By comparing experiments 8 and 10, we see that identical parameters are noticeably more compressed with the addition of the two methods. However, by comparing experiments 9-11 with 12-19 we see the most impactful choice was the decision to apply SVD more prominently across the time-axis, forgoing the width entirely ($\frac{1}{1215}$ represents 1 column of the $1215px$ width.). Similarly, reducing the k-value significantly improved compression as well but only to a certain point, after which the increased density of the the error array more than made up for the decreased size of the truncated SVD arrays. The improvement in accuracy at smaller values of k is likely due to the quantization of E meaning more error terms were less impactful on the final size of the compressed dataset.

2.5 Other Epsilon Values

Lastly, though not shown here, experiments were run on each of the other expected epsilon values ($10^{-1}, 10^{-3}, 10^{-4}$). Using identical SVD kernel size as the epsilon 10^-2 case, we simply modified the value of k to maximize compression.

In the case of epsilon 10^{-1}, dropping k to 10 or smaller was effective in reducing the size of the SVD arrays, while the higher error threshold meant that the error array was naturally more sparse, leading to a compression ratio between 30-40.

On the other hand, in both the $\epsilon = 10^{-3}$ and $\epsilon = 10^{-4}$ cases, the error was so great that even with large k values, a majority of terms were outside the error threshold, leaving E densely populated, and leading to a compression ratio less than one. To combat this, we considered the opposite case. By allowing k to be very small ($k < 10$) we ensure that the SVD matrices have very small relative size. As a result, more than 90% of the values in D' needed error correction. However, by defining E to be a dense array, we see that a fully populated E, will have roughly a quarter or a half the size of the original data (depending on if int8 or int16 types are used during quantization). Considering that the size of U, S and V are negligible with small k, this method allows us to achieve a compression ratio slightly less than the compression of quantization alone.

In particular, for $\epsilon = 10^{-3}$ a compression ratio of $\rho = 3.85$ was achieved by quantizing the errors to type uint8 with $k = 2$, and in the $\epsilon = 10^{-4}$ case a compression ratio of $\rho = 1.9$ was achieved by quantizing the errors to type uint16 with $k = 5$. uint16 was chosen in the second case because by the formulation of quantization described in (2.3), the range of errors was too great to linearly quantize in a 8 bit range.

3 Future Improvements

Future works looking to improve further on these results should implement two additional features which we were unable to. First, quantizing in the SVD arrays U, S and V has the potential to further reduce the memory footprint of the data. Moreover, with the use of a more powerful computer (and more efficiently implemented code) we encourage future works to attempt to compress larger subsets of the data. As we see from experiments 19 and 21, there is a huge jump in efficiency by expanding the SVD size, however, the resulting 4000x(855*3) kernel took approximately 15s per SVD cycle and 20 minutes to compress the whole dataset. Moreover, experiments using even larger matrices encountered run-times taking numerous hours due to the superlinear scaling of SVD. These larger SVD kernels are likely to further take advantage of spatial similarities within the data and allow for further compression, so we highly recommend these steps to all future works on the subject.

4 Conclusion

Our exploration of low-rank SVD approximation for compressing the Red Sea re-analysis dataset has highlighted the efficacy of techniques like data zero-centering and selective error quantization. These methods significantly enhance the expressiveness of the SVD representation, achieving higher compression ratios while adhering to an acceptable error threshold. Experimentation revealed that the choice of tiling dimensions and SVD rank are critical factors in balancing memory efficiency and computational feasibility, and we achieved a competitive compression ratio of 17.8 when relative error was bounded at 10^{-2}. However, challenges remain, such as the need for further optimization of tiling strategies and handling quantization for all components without compromising accuracy. Future work could explore alternative decomposition methods, adaptive rank selection, and broader applications of the proposed compression framework. This study underscores the potential of low-rank approximations to address storage challenges in large-scale scientific datasets.

References

[1] Hari Dasari, Yesubabu V, Sabique Langodan, Y. Abualnaja, Srinivas Desamsetti, Vankayalapati Koteswararao, Luong Thang, and Ibrahim Hoteit. High-resolution climate characteristics of the arabian gulf based on a validated regional reanalysis. *Meteorological Applications*, 29, 10 2022.
[2] TileDB, Inc. Tiledb: Universal storage engine. https://tiledb.com, 2024.
[3] Lloyd N. Trefethen and III David Bau. *Numerical Linear Algebra.* SIAM: Society for Industrial and Applied Mathematics, Philadelphia, PA, 1997.