Learning-Based Low-Rank Approximations

Piotr Indyk
CSAIL, MIT
`indyk@mit.edu`

Ali Vakilian[*]
University of Wisconsin-Madison
`vakilian@wisc.edu`

Yang Yuan[*]
Tsinghua University
`yuanyang@tsinghua.edu.cn`

Abstract

We introduce a "learning-based" algorithm for the *low-rank decomposition* problem: given an $n \times d$ matrix A, and a parameter k, compute a rank-k matrix A' that minimizes the approximation loss $\|A - A'\|_F$. The algorithm uses a training set of input matrices in order to optimize its performance. Specifically, some of the most efficient approximate algorithms for computing low-rank approximations proceed by computing a projection SA, where S is a sparse random $m \times n$ "sketching matrix", and then performing the singular value decomposition of SA. We show how to replace the random matrix S with a "learned" matrix of the same sparsity to reduce the error.

Our experiments show that, for multiple types of data sets, a learned sketch matrix can substantially reduce the approximation loss compared to a random matrix S, sometimes by one order of magnitude. We also study mixed matrices where only some of the rows are trained and the remaining ones are random, and show that matrices still offer improved performance while retaining worst-case guarantees.

1 Introduction

The success of modern machine learning made it applicable to problems that lie outside of the scope of "classic AI". In particular, there has been a growing interest in using machine learning to improve the performance of "standard" algorithms, by fine-tuning their behavior to adapt to the properties of the input distribution, see e.g., [1–13]. This "learning-based" approach to algorithm design has attracted a considerable attention over the last few years, due to its potential to significantly improve the efficiency of some of the most widely used algorithmic tasks. Many applications involve processing streams of data (video, data logs, customer activity etc) by executing the same algorithm on an hourly, daily or weekly basis. These data sets are typically not "random" or "worst-case"; instead, they come from some distribution which does not change rapidly from execution to execution. This makes it possible to design better algorithms tailored to the specific data distribution, trained on past instances of the problem.

The method has been particularly successful in the context of *compressed sensing*. In the latter framework, the goal is to recover an approximation to an n-dimensional vector x, given its "linear measurement" of the form Sx, where S is an $m \times n$ matrix. Theoretical results [14, 15] show that, if the matrix S is selected *at random*, it is possible to recover the k largest coefficients of x with high probability using a matrix S with $m = O(k \log n)$ rows. This guarantee is general and applies to arbitrary vectors x. However, if vectors x are selected from some natural distribution (e.g., they represent images), recent works [8, 9, 11] show that one can use samples from that distribution to compute matrices S that improve over a completely random matrix in terms of the recovery error.

Compressed sensing is an example of a broader class of problems which can be solved using random projections. Another well-studied problem of this type is *low-rank decomposition*: given an $n \times d$ matrix A, and a parameter k, compute a rank-k matrix $[A]_k = \text{argmin}_{A':\ \text{rank}(A') \le k} \|A - A'\|_F$.

Low-rank approximation is one of the most widely used tools in massive data analysis, machine learning and statistics, and has been a subject of many algorithmic studies. In particular, multiple

[*]This work was mostly done when the second and third authors were at MIT.

algorithms developed over the last decade use the "sketching" approach, see e.g., [16–24]. Its idea is to use efficiently computable random projections (a.k.a., "sketches") to reduce the problem size before performing low-rank decomposition, which makes the computation more space and time efficient. For example, [16, 19] show that if S is a random matrix of size $m \times n$ chosen from an appropriate distribution, for m depending on ϵ, then one can recover a rank-k matrix A' such that

$$\|A - A'\|_F \leq (1+\epsilon)\|A - [A]_k\|_F$$

by performing an SVD on $SA \in \mathbb{R}^{m\times d}$ followed by some post-processing. Typically the sketch length m is small, so the matrix SA can be stored using little space (in the context of streaming algorithms) or efficiently communicated (in the context of distributed algorithms). Furthermore, the SVD of SA can be computed efficiently, especially after another round of sketching, reducing the overall computation time. See the survey [25] for an overview of these developments.

In light of the aforementioned work on learning-based compressive sensing, it is natural to ask whether similar improvements in performance could be obtained for other sketch-based algorithms, notably for low-rank decompositions. In particular, reducing the sketch length m while preserving its accuracy would make sketch-based algorithms more efficient. Alternatively, one could make sketches more accurate for the same values of m. This is the problem we address in this paper.

Our Results. Our main finding is that learned sketch matrices can indeed yield (much) more accurate low-rank decompositions than purely random matrices. We focus our study on a streaming algorithm for low-rank decomposition due to [16, 19], described in more detail in Section 2. Specifically, suppose we have a training set of matrices $\mathsf{Tr} = \{A_1, \ldots, A_N\}$ sampled from some distribution D. Based on this training set, we compute a matrix S^* that (locally) minimizes the empirical loss

$$\sum_i \|A_i - \mathrm{SCW}(S^*, A_i)\|_F \tag{1}$$

where $\mathrm{SCW}(S^*, A_i)$ denotes the output of the aforementioned Sarlos-Clarkson-Woodruff streaming low-rank decomposition algorithm on matrix A_i using the sketch matrix S^*. Once the the sketch matrix S^* is computed, it can be used instead of a random sketch matrix in all future executions of the SCW algorithm.

We demonstrate empirically that, for multiple types of data sets, an optimized sketch matrix S^* can substantially reduce the approximation loss compared to a random matrix S, sometimes by one order of magnitude (see Figure 1). Equivalently, the optimized sketch matrix can achieve the same approximation loss for lower values of m.

A possible disadvantage of learned sketch matrices is that an algorithm that uses them no longer offers *worst-case* guarantees. As a result, if such an algorithm is applied to an input matrix that does not conform to the training distribution, the results might be worse than if random matrices were used. To alleviate this issue, we also study *mixed* sketch matrices, where (say) half of the rows are trained and the other half are random. We observe that if such matrices are used in conjunction with the SCW algorithm, its results are no worse than if only the random part of the matrix was used[2]. Thus, the resulting algorithm inherits the worst-case performance guarantees of the random part of the sketching matrix. At the same time, we show that mixed matrices still substantially reduce the approximation loss compared to random ones, in some cases nearly matching the performance of "pure" learned matrices with the same number of rows. Thus, mixed random matrices offer "the best of both worlds": improved performance for matrices from the training distribution, and worst-case guarantees otherwise.

2 Preliminaries

Notation. Consider a distribution D on matrices $A \in \mathbb{R}^{n\times d}$. We define the training set as $\{A_1, \cdots, A_N\}$ sampled from D. For matrix A, its *singular value decomposition (SVD)* can be written as $A = U\Sigma V^\top$ such that both U and V have *orthonormal columns* and $\Sigma = \mathrm{diag}\{\lambda_1, \cdots, \lambda_d\}$ is a diagonal matrix with nonnegative entries. In many applications it is quicker and more economical to

[2]We note that this property is non-trivial, in the sense that it does not hold for *all* sketching algorithms. The proof of this property for Algorithm 1 is deferred to the longer version of this paper.

Algorithm 1 Rank-k approximation of a matrix A using a sketch matrix S (from Section 4.1.1 of [19])

```
1: Input: $A \in \mathbb{R}^{n\times d}, S \in \mathbb{R}^{m\times n}$
2: $U, \Sigma, V^\top \leftarrow$ Compact-SVD($SA$)    ▷ $\{r = \text{rank}(SA), U \in \mathbb{R}^{m\times r}, V \in \mathbb{R}^{d\times r}\}$
3: Return: $[AV]_k V^\top$
```

compute the *compact SVD* which only contains the rows and columns corresponding to the non-zero singular values of Σ: $A = U^c\Sigma^c(V^c)^\top$ where $U^c \in \mathbb{R}^{n\times r}$, $\Sigma^c \in \mathbb{R}^{r\times r}$ and $V^c \in \mathbb{R}^{d\times r}$.

How sketching works. We start by describing the SCW algorithm (Algorithm 1) for low-rank approximation. The algorithm computes the $\text{SVD}(SA) := U\Sigma V^\top$, and compute the best rank-k approximation of AV. Finally it outputs $[AV]_k V^\top$ as a rank-k approximation of A. Note that if m is much smaller than d and n, the space bound of this algorithm is significantly better than when computing a rank-k approximation for A in the naïve way. Thus, minimizing m automatically reduces the space usage of the algorithm.

Sketching matrix. We use matrix S that is sparse. Specifically, each column of S has exactly one non-zero entry, which is either $+1$ or -1. This means that the fraction of non-zero entries in S is $1/m$. Therefore, one can use a vector to represent S, which is very memory efficient. It is worth noting, however, after multiplying S with other matrices, the resulting matrix is in general not sparse.

3 Training Algorithm

In this section, we describe our learning-based algorithm for computing a data dependent sketch S. The main idea is to use backpropagation algorithm to compute the stochastic gradient of S with respect to the rank-k approximation loss in Equation 1, where the initial value of S is the same random sparse matrix used in SCW. Once we have the stochastic gradient, we can run stochastic gradient descent (SGD) algorithm to optimize S, in order to improve the loss. Our algorithm maintains the sparse structure of S, and only optimizes the values of the n non-zero entries (initially $+1$ or -1).

However, the standard SVD implementation (step 2 in Algorithm 1) is not differentiable, which means we cannot get the gradient in the straightforward way. To make SVD implementation differentiable, we use the fact that the SVD procedure can be represented as m individual top singular value decompositions (see e.g. [26]), and that every top singular value decomposition can be computed using the power method. The full description is deferred to the long version of this paper.

Due to the extremely long computational chain, it is infeasible to write down the explicit form of loss function or the gradients. However, just like how modern deep neural networks compute their gradients, we used the autograd feature in PyTorch to *numerically* compute the gradient with respect to the sketching matrix S.

We emphasize again that our method is only optimizing S for the training phase. After S is fully trained, we still call Algorithm 1 for low rank approximation, which has exactly the same running time as the SCW algorithm, but with better performance.

4 Experimental Results

The main question considered in this paper is whether, for natural matrix datasets, optimizing the sketch matrix S can improve the performance of the sketching algorithm for the low-rank decomposition problem. To answer this question, we implemented and compared the following methods for computing $S \in \mathbb{R}^{m\times n}$.

- **Sparse Random**. Sketching matrices are generated at random as in [20]. Specifically, we select a random hash function $h : [n] \to [m]$, and for all $i \in [n]$, $S_{h[i],i}$ is selected to be either $+1$ or -1 with equal probability. All other entries in S are set to 0.
- **Dense Random**. All entries in the sketching matrices are sampled from Gaussian distribution.
- **Learned**. Using the sparse random matrix as the initialization, we optimize the sketching matrix using the training set, and return the optimized matrix.

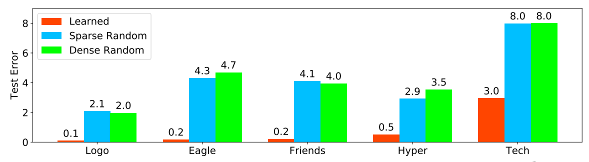

Figure 1: Test error by datasets and sketching matrices For $k = 10, m = 20$

- **Mixed (J)**. We first generate two sparse random matrices $S_1, S_2 \in \mathbb{R}^{\frac{m}{2} \times n}$ (assuming m is even), and define S to be their combination. We then optimize S using the training set, but only S_1 will be updated, while S_2 is fixed. Therefore, S is a mixture of learned matrix and random matrix, and the first matrix is trained *jointly* with the second one.
- **Mixed (S)**. We first compute a learned matrix $S_1 \in \mathbb{R}^{\frac{m}{2} \times n}$ using the training set, and then append another sparse random matrix S_2 to get $S \in \mathbb{R}^{m \times n}$. Therefore, S is a mixture of learned matrix and random matrix, but the learned matrix is trained *separately*.

Datasets. We used a variety of datasets to test the performance of our methods:

- **Videos[3]: Logo, Friends, Eagle.** We downloaded three high resolution videos from Youtube, including logo video, Friends TV show, and eagle nest cam. From each video, we collect 500 frames of size $1920 \times 1080 \times 3$ pixels, and use 400 (100) matrices as the training (test) set. For each frame, we resize it as a 5760×1080 matrix.
- **Hyper.** We use matrices from HS-SOD, a dataset for hyperspectral images from natural scenes [27]. Each matrix has 1024×768 pixels, and we use 400 (100) matrices as the training (test) set.
- **Tech.** We use matrices from TechTC-300, a dataset for text categorization [28]. Each matrix has $835,422$ rows, but on average only $25,389$ of the rows contain non-zero entries. On average each matrix has 195 columns. We use 200 (95) matrices as the training (test) set.

Evaluation metric. To evaluate the quality of a sketching matrix S, it suffices to evaluate the output of Algorithm 1 using the sketching matrix S on different input matrices A. For a collection of matrices Te, we define the error of the sketch S as $\text{Err}(\mathsf{Te}, S) \triangleq \mathbf{E}_{A\sim\mathsf{Te}}\|A - \text{SCW}(S, A)\|_F - \mathbf{E}_{A\sim\mathsf{Te}}\|A - [A]_k\|_F$, where the second term denotes the optimal approximation loss on Te.

In our datasets, some of the matrices have much larger singular values than the others. To avoid imbalance in the dataset, we normalize the matrices so that their top singular values are all equal.

Table 1: Test error in various settings

k, m, Sketch	Logo	Eagle	Friends	Hyper	Tech
$10, 20$, Learned	0.10	0.18	0.22	0.52	2.95
$10, 20$, Random	2.09	4.31	4.11	2.92	7.99
$20, 20$, Learned	0.61	0.66	1.41	1.68	7.79
$20, 20$, Random	4.18	5.79	9.10	5.71	14.55
$20, 40$, Learned	0.18	0.41	0.42	0.72	3.09
$20, 40$, Random	1.19	3.50	2.44	2.23	6.20
$30, 30$, Learned	0.72	1.06	1.78	1.90	7.14
$30, 30$, Random	3.11	6.03	6.27	5.23	12.82

Table 2: Comparison with mixed sketches

k, m, Sketch	Logo	Hyper	Tech
$10, 20$, Learned	0.10	0.52	2.95
$10, 20$, Mixed (J)	0.20	0.78	3.73
$10, 20$, Mixed (S)	0.24	0.87	3.69
$10, 20$, Random	2.09	2.92	7.99
$10, 40$, Learned	0.04	0.28	1.16
$10, 40$, Mixed (J)	0.05	0.34	1.31
$10, 40$, Mixed (S)	0.05	0.34	1.20
$10, 40$, Random	0.45	1.12	3.28

We first test all methods on different datasets, with various combination of k, m. See Figure 1 for the results when $k = 10, m = 20$. As we can see, for video datasets, learned sketching matrices can get $20\times$ better test error than the sparse random or dense random sketching matrices. For other datasets, learned sketching matrices are still more than $2\times$ better. We also include the test error results in Table 1 for the case when $k = 20, 30$.

In Table 2, we investigate the performance of the mixed sketching matrices by comparing them with random and learned sketching matrices. In all scenarios, mixed sketching matrices yield much better results than random sketching matrices, and sometimes the results are comparable to those of learned sketching matrices. This means, in most cases it suffices to train one half of the sketching matrix to obtain good empirical results, and at the same time, we can use the remaining random half of the sketch matrix to obtain worst-case guarantees.

[3]They can be downloaded from `http://youtu.be/L5HQoFIaT4I`, `http://youtu.be/xmLZsEfXEgE` and `http://youtu.be/ufnf_q_3Ofg`

References

[1] Jun Wang, Wei Liu, Sanjiv Kumar, and Shih-Fu Chang. Learning to hash for indexing big data - a survey. *Proceedings of the IEEE*, 104(1):34–57, 2016.

[2] Elias Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. In *Advances in Neural Information Processing Systems*, pages 6348–6358, 2017.

[3] Tim Kraska, Alex Beutel, Ed H Chi, Jeffrey Dean, and Neoklis Polyzotis. The case for learned index structures. In *Proceedings of the 2018 International Conference on Management of Data*, pages 489–504, 2018.

[4] Maria-Florina Balcan, Travis Dick, Tuomas Sandholm, and Ellen Vitercik. Learning to branch. In *International Conference on Machine Learning*, pages 353–362, 2018.

[5] Thodoris Lykouris and Sergei Vassilvitskii. Competitive caching with machine learned advice. In *International Conference on Machine Learning*, pages 3302–3311, 2018.

[6] Manish Purohit, Zoya Svitkina, and Ravi Kumar. Improving online algorithms via ml predictions. In *Advances in Neural Information Processing Systems*, pages 9661–9670, 2018.

[7] Michael Mitzenmacher. A model for learned bloom filters and optimizing by sandwiching. In *Advances in Neural Information Processing Systems*, pages 464–473, 2018.

[8] Ali Mousavi, Ankit B Patel, and Richard G Baraniuk. A deep learning approach to structured signal recovery. In *Communication, Control, and Computing (Allerton), 2015 53rd Annual Allerton Conference on*, pages 1336–1343. IEEE, 2015.

[9] Luca Baldassarre, Yen-Huan Li, Jonathan Scarlett, Baran Gözcü, Ilija Bogunovic, and Volkan Cevher. Learning-based compressive subsampling. *IEEE Journal of Selected Topics in Signal Processing*, 10(4):809–822, 2016.

[10] Ashish Bora, Ajil Jalal, Eric Price, and Alexandros G Dimakis. Compressed sensing using generative models. In *International Conference on Machine Learning*, pages 537–546, 2017.

[11] Chris Metzler, Ali Mousavi, and Richard Baraniuk. Learned d-amp: Principled neural network based compressive image recovery. In *Advances in Neural Information Processing Systems*, pages 1772–1783, 2017.

[12] Paul Hand and Vladislav Voroninski. Global guarantees for enforcing deep generative priors by empirical risk. In *Conference On Learning Theory*, 2018.

[13] Chen-Yu Hsu, Piotr Indyk, Dina Katabi, and Ali Vakilian. Learning-based frequency estimation algorithms. *International Conference on Learning Representations*, 2019.

[14] David L Donoho. Compressed sensing. *IEEE Transactions on information theory*, 52(4):1289–1306, 2006.

[15] Emmanuel J Candès, Justin Romberg, and Terence Tao. Robust uncertainty principles: Exact signal reconstruction from highly incomplete frequency information. *IEEE Transactions on information theory*, 52(2):489–509, 2006.

[16] Tamas Sarlos. Improved approximation algorithms for large matrices via random projections. In *47th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 143–152, 2006.

[17] Franco Woolfe, Edo Liberty, Vladimir Rokhlin, and Mark Tygert. A fast randomized algorithm for the approximation of matrices. *Applied and Computational Harmonic Analysis*, 25(3):335–366, 2008.

[18] Nathan Halko, Per-Gunnar Martinsson, and Joel A Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM review*, 53(2):217–288, 2011.

[19] Kenneth L Clarkson and David P Woodruff. Numerical linear algebra in the streaming model. In *Proceedings of the forty-first annual symposium on Theory of computing (STOC)*, pages 205–214, 2009.

[20] Kenneth L Clarkson and David P Woodruff. Low-rank approximation and regression in input sparsity time. *Journal of the ACM (JACM)*, 63(6):54, 2017.

[21] Jelani Nelson and Huy L Nguyên. Osnap: Faster numerical linear algebra algorithms via sparser subspace embeddings. In *Foundations of Computer Science (FOCS), 2013 IEEE 54th Annual Symposium on*, pages 117–126, 2013.

[22] Xiangrui Meng and Michael W Mahoney. Low-distortion subspace embeddings in input-sparsity time and applications to robust linear regression. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 91–100, 2013.

[23] Christos Boutsidis and Alex Gittens. Improved matrix algorithms via the subsampled randomized hadamard transform. *SIAM Journal on Matrix Analysis and Applications*, 34(3):1301–1340, 2013.

[24] Michael B Cohen, Sam Elder, Cameron Musco, Christopher Musco, and Madalina Persu. Dimensionality reduction for k-means clustering and low rank approximation. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, pages 163–172, 2015.

[25] David P Woodruff. Sketching as a tool for numerical linear algebra. *Foundations and Trends® in Theoretical Computer Science*, 10(1–2):1–157, 2014.

[26] Zeyuan Allen-Zhu and Yuanzhi Li. Lazysvd: even faster svd decomposition yet without agonizing pain. In *Advances in Neural Information Processing Systems*, pages 974–982, 2016.

[27] Nevrez Imamoglu, Yu Oishi, Xiaoqiang Zhang, Guanqun Ding, Yuming Fang, Toru Kouyama, and Ryosuke Nakamura. Hyperspectral image dataset for benchmarking on salient object detection. In *Tenth International Conference on Quality of Multimedia Experience, (QoMEX)*, pages 1–3, 2018.

[28] Dmitry Davidov, Evgeniy Gabrilovich, and Shaul Markovitch. Parameterized generation of labeled datasets for text categorization based on a hierarchical directory. In *Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '04, pages 250–257, 2004.