Polynomial Tensor Sketch
for Element-wise Function of Low-Rank Matrix

Insu Han, Haim Avron, Jinwoo Shin

1School of Electrical Engineering, Korea Advanced Institute of Science and Technology
2Department of Applied Mathematics, Tel Aviv University
{insu.han,jinwoos}@kaist.ac.kr, haimav@tauex.tau.ac.il

Abstract

This paper studies how to sketch element-wise functions of low-rank matrices. Formally, given low-rank matrix $A = [A_{ij}]$ and scalar non-linear function $f$, we aim for finding an approximated low-rank representation of (high-rank) matrix $f(A_{ij})$. To this end, we propose an efficient sketch algorithm whose complexity is significantly lower than the number of entries of $A$, i.e., it runs without accessing all entries of $f(A_{ij})$ explicitly. Our main idea is to combine a polynomial approximation on $f$ with the existing tensor sketch scheme approximating monomials of entries of $A$. To balance errors of the two approximation components in an optimal manner, we address a novel regression formula to find polynomial coefficients given $A$ and $f$. We demonstrate the applicability and superiority of the proposed scheme under the tasks of kernel SVM classification and optimal transport.

1 Introduction

Given a low-rank matrix $A = [A_{ij}] \in \mathbb{R}^{n \times n}$ with $A = UV^T$ for some matrices $U, V \in \mathbb{R}^{n \times d}$ with $d \ll n$ and a scalar non-linear function $f: \mathbb{R} \rightarrow \mathbb{R}$, we are interested in the following element-wise matrix function:

$$f \odot (A) \in \mathbb{R}^{n \times n}, \quad \text{where } f \odot (A)_{ij} := [f(A_{ij})].$$

Our goal is to design a fast algorithm computing small (or thin) matrices $T_U, T_V$ in time $o(n^2)$, e.g., $O(n)$, such that $f \odot (A) \approx T_U T_V^T$. Namely, it should run without computing all entries of $A$ or $f \odot (A)$ explicitly. This can lead to an $o(n^2)$-time approximation scheme of $f \odot (A)x \approx T_U T_V^T x$ for an arbitrary vector $x \in \mathbb{R}^n$ due to the associative property of matrix multiplication, where the exact computation of $f \odot (A)x$ requires the complexity of $O(n^3)$.

The matrix-vector multiplication $f \odot (A)x$ or the low-rank decomposition $f \odot (A) \approx T_U T_V^T$ are useful in many machine learning algorithms. For example, the Gram matrices of certain kernel functions, e.g., polynomial and radial basis function (RBF), are element-wise matrix functions where the rank is the dimension of the underlying data. Such matrices are the cornerstone of so-called kernel methods and the ability to multiply the Gram matrix to a vector suffices for most kernel learning. The Sinkhorn-Knopp algorithm is a powerful, yet simple, tool for computing optimal transport distances [7, 2] and also involves the matrix-vector multiplication $f \odot (A)x$ with $f(x) = \exp(x)$. Finally, $f \odot (UV^T)$ can also describe the non-linear computation of activation in a layer of deep neural networks [9], where $U, V$ and $f$ correspond to input, weight and activation function (e.g., sigmoid or ReLU) of the previous layer, respectively.

*In this paper, we primarily focus on the square matrix $A$ for simplicity, but it is straightforward to extend our results to the case of non-square matrices.
Unlike the element-wise matrix function $f^\odot(A)$, a traditional matrix function $f(A)$ is defined on their eigenvalues [13] and possess clean algebraic properties, i.e., it preserves eigenvectors. For example, given a diagonalizable matrix $A = PDP^{-1}$, it is defined that $f(A) = Pf^\odot(D)P^{-1}$. A classical problem addressed in the literature is of approximating the trace of $f(A)$ efficiently [11, 21, 10]. However, these methods are not applicable to our problem because element-wise matrix functions are fundamentally different from the traditional function of matrices, e.g., they do not guarantee the spectral property. To the best of our knowledge, we are unaware of any approximation algorithm that targets general element-wise matrix functions. On this line, Random Fourier feature (RFF) [17, 15] and Tensorsketch [16] only approximate special classes of element-wise matrix functions of $f(x) = \exp(x)$ and $f(x) = x^k$, respectively. We aim for not only designing an approximation framework for general $f$, but also outperforming RFF even for the special case $f(x) = \exp(x)$.

To this end, our high-level idea is to combine two approximation schemes: Tensorsketch approximating the element-wise matrix function of monomial $x^k$ and a degree-$r$ polynomial approximation $p_r(x) = \sum_{j=0}^{r} c_j x^j$ of function $f$, e.g., $f(x) = \exp(x) \approx p_r(x) = \sum_{j=0}^{r} \frac{1}{j!} x^j$. More formally, we consider the following approximation:

$$f^\odot(A) \approx \sum_{j=0}^{r} c_j \odot (x^j)^\odot(A) \approx \sum_{j=0}^{r} c_j \odot \text{Tensorsketch}\left((x^j)^\odot(A)\right),$$

which we call POLY-Tensorsketch. This is a linear-time approximation scheme with respect to $n$ under the choice of $r = \mathcal{O}(1)$. Here, a non-trivial challenge occurs: a larger degree $r$ is required to approximate an arbitrary function $f$ better, while the approximation error of Tensorsketch is known to increase exponentially with respect to $r$ [16, 3]. Hence, it is important to choose good $c_j$’s for balancing two approximation errors in both (a) and (b). The known (truncated) polynomial expansions such as Taylor or Chebyshev [14] are far from being optimal for this purpose (see Section 3.2 and 4.1 for more details).

To tackle the challenge, we address it as an optimization task in $c_j$’s for minimizing the approximation error of POLY-Tensorsketch. However, the exact optimization is intractable since the objective involves an expectation taken over random variables whose supports are exponentially large. Instead, we derive a novel tractable upper bound to optimize which turns out to be a generalized ridge regression [12] that has a closed-form solution. It indeed regularizes coefficients to be exponentially decaying to compensate the exponentially growing errors of Tensorsketch, while simultaneously maintaining a good polynomial approximation to the scalar function $f$ given entries of $A$. We further reduce its complexity by regressing only a subset of entries of the matrix. Finally, we construct the regression with respect to Chebyshev polynomial basis (instead of monomials), i.e., $p_r(x) = \sum_{j=0}^{r} c_j t_j(x)$ for the Chebyshev polynomial $t_j(x)$ of degree $j$, to resolve a numerical issue under a large degree.

We evaluate the approximation quality of our algorithm under the RBF kernels of synthetic and real-world datasets. Then, we apply the proposed method to two machine learning tasks: classification using kernel SVM [6, 19] and computation of optimal transport distances [7] that require to compute element-wise matrix functions with $f(x) = \exp(x)$. Our experimental results confirm that our scheme is at the order of magnitude faster than the exact method with a marginal loss on accuracy. Furthermore, our scheme also significantly outperforms a state-of-the-art approximation method, RFF for the aforementioned applications.

2 Preliminaries

In this section, we provide backgrounds for randomized sketching algorithms, i.e., CountSketch and Tensorsketch, which are key components of the proposed scheme for approximating element-wise matrix functions. CountSketch [5, 23] was proposed for an effective dimensionality reduction of high-dimensional vector $u \in \mathbb{R}^d$. Formally, consider a random hash function $h : [d] \rightarrow [m]$ and a random sign function $s : [d] \rightarrow \{-1, +1\}$, where $[d] := \{1, 2, \ldots, d\}$. Then, CountSketch transforms $u$ into $C_u \in \mathbb{R}^m$ such that $[C_u]_j := \sum_{i : h(i) = j} s(i) u_i$ for $j \in [m]$. The algorithm takes $\mathcal{O}(d)$ time to run since it requires a single pass over the input. It is known that applying the same $^1$ CountSketch transform on two vectors preserves the dot-product, i.e., $\langle u, v \rangle = E[\langle C_u, C_v \rangle]$.

$^1$Use the same hash and sign functions.
Algorithm 1 \textsc{TensorSketch} [16]

1: \textbf{Input}: matrix $U \in \mathbb{R}^{n \times d}$, degree $k$ and sketch dimension $m$
2: Draw $k$ i.i.d. random hash functions $h_1, \ldots, h_k : [d] \to [m]$
3: Draw $k$ i.i.d. random sign functions $s_1, \ldots, s_k : [d] \to \{-1, +1\}$
4: $C_{U}^{(i)} \leftarrow \text{\textsc{CountSketch}}$ on each row of $U$ using $h_i$ and $s_i$ for $i = 1, \ldots, k$.
5: $T_{U}^{(k)} \leftarrow \text{\textsc{FFT}}^{-1} \left( \text{\textsc{FFT}}(C_{U}^{(1)}) \otimes \cdots \otimes \text{\textsc{FFT}}(C_{U}^{(k)}) \right)$
6: \textbf{Output}: $T_{U}^{(k)}$

In the above algorithm, $\otimes$ is the element-wise multiplication (also called the Hadamard product) between two matrices of the same dimension and $\text{\textsc{FFT}}(U)$, $\text{\textsc{FFT}}^{-1}(U)$ are the fast Fourier transform and its inverse applied to each row of a matrix $U \in \mathbb{R}^{n \times d}$, respectively, where they take time $O(nd \log d)$ to compute. Then, one can check that Algorithm 1 runs in time $O(nk(d + m \log m))$ because it requires to run $\text{\textsc{FFT}}$ and $\text{\textsc{CountSketch}}$ $k$ times. Recently, Avron et al. [3] proved a tight bound on the variance (or error) of $\text{\textsc{TensorSketch}}$ as follows.

**Theorem 1 (Avron et al. [3])** Given $U, V \in \mathbb{R}^{n \times d}$, let $T_{U}^{(k)}, T_{V}^{(k)} \in \mathbb{R}^{n \times m}$ be the same $\text{\textsc{TensorSketch}}$ of $U, V$ with degree $k \geq 0$ and sketch dimension $m$. Then, it holds that

$$E \left[ \left\| (UV^\top)^{\otimes k} - T_{U}^{(k)}T_{V}^{(k)^\top} \right\|_F^2 \right] \leq \frac{(2 + 3^k) \left( \sum_i \langle \sum_j U_{ij}^2 \rangle^k \right) \left( \sum_i \langle \sum_j V_{ij}^2 \rangle^k \right)}{m}.$$  

(1)

The above theorem implies that the error of $\text{\textsc{TensorSketch}}$ becomes small for large sketch dimension $m$, but can exponentially grow with respect to degree $k$.

3 Linear-time approximation of element-wise matrix functions

Given a scalar function $f : \mathbb{R} \to \mathbb{R}$ and matrices $U, V \in \mathbb{R}^{n \times d}$ with $d \ll n$, we attempt to design an efficient algorithm to find $T_U, T_V \in \mathbb{R}^{n \times d'}$ with $d' \ll n$ in time $o(n^2)$ such that

$$f^{\otimes}(UV^\top) \approx T_U T_V^\top \in \mathbb{R}^{n \times n}.$$  

Namely, we aim for finding a low-rank approximation of $f^{\otimes}(UV^\top)$ without computing all $n^2$ entries of $UV^\top$. We first describe the proposed approximation scheme in Section 3.1 and provide further ideas for minimizing the approximation gap in Section 3.2.

\footnote{Unless stated otherwise, we define $T^{(0)}_u := 1.$}
3.1 Poly-TensorSketch

Suppose we have a polynomial $p_r(x) = \sum_{j=0}^r c_j x^j$ approximating $f(x)$, e.g., $f(x) = \exp(x)$ and its (truncated) Taylor series $p_r(x) = \sum_{j=0}^r \frac{1}{j!}x^j$. Then, we consider the following approximation scheme, coined Poly-TensorSketch:

$$f^\circ(UV^\top) \approx \sum_{j=0}^r c_j (UV^\top)^\otimes j \approx \sum_{j=0}^r c_j T_U^{(j)} T_V^{(j)^\top}, \quad (2)$$

where $T_U^{(j)}, T_V^{(j)} \in \mathbb{R}^{n \times m}$ are the same TensorSketch of $U, V$ with degree $j$ and sketch dimension $m$, respectively. Namely, our main idea is to combine (a) a polynomial approximation of a scalar function with (b) the randomized tensor sketch of a matrix. Instead of running Algorithm 1 independently for each $j \in [r]$, we utilize the following recursive relation to amortize operations:

$$T_U^{(j)} = \text{FFT}^{-1}\left(\text{FFT}(C_U) \otimes \text{FFT}(T_U^{(j-1)})\right),$$

where $C_U$ is COUNTSketch on each row of $U$ whose randomness is independently drawn from that of $T_U^{(j-1)}$. Since each recursive step can be computed in time $O(n(d + m \log m))$, computing all $T_U^{(j)}$ for $j \in [r]$ requires $O(nr(d + m \log m))$ operations. Hence, the overall complexity of Poly-TensorSketch, formally described in Algorithm 2, is $O(n)$ if $r, d, m = O(1)$.

**Algorithm 2 Poly-TensorSketch**

1: **Input**: $U, V \in \mathbb{R}^{n \times d}$, degree $r$, coefficients $c_0, \ldots, c_r$, sketch dimension $m$
2: Draw $r$ independent random hash functions $h_1, \ldots, h_r : [d] \to [m]$
3: Draw $r$ independent random sign functions $s_1, \ldots, s_r : [d] \to \{1, -1\}$
4: $T_U^{(0)}, T_V^{(0)} \leftarrow \text{COUNTSketch}$ of $U, V$ using $h_1$ and $s_1$, respectively.
5: $F_U, F_V \leftarrow \text{FFT}(T_U^{(0)}), \text{FFT}(T_V^{(0)})$
6: $\Gamma \leftarrow c_0 T_U^{(0)} T_V^{(0)^\top} + c_1 T_U^{(1)} T_V^{(1)^\top}$
7: for $j = 2$ to $r$ do
8: $C_U, C_V \leftarrow \text{COUNTSketch}$ of $U, V$ using $h_j$ and $s_j$, respectively.
9: $F_U, F_V \leftarrow \text{FFT}(C_U) \odot F_U, \text{FFT}(C_V) \odot F_V$
10: $T_U^{(j)}, T_V^{(j)} \leftarrow \text{FFT}^{-1}(F_U), \text{FFT}^{-1}(F_V)$
11: $\Gamma \leftarrow \Gamma + c_j T_U^{(j)} T_V^{(j)^\top}$
12: end for
13: **Output**: $\Gamma$

Observe that multiplication of $\Gamma$ (i.e., the output of Algorithm 2) and an arbitrary vector $x \in \mathbb{R}^d$ can be done in time $O(nmr)$ due to $\Gamma x = \sum_{j=0}^r c_j T_U^{(j)} T_V^{(j)^\top} x$. Hence, for $r, m, d = O(1)$, Poly-TensorSketch can approximate $f^\circ(UV^\top)x$ in time $O(nr(d + m \log m)) = O(n)$, where we prove the following error bound.

**Proposition 1** Given $U, V \in \mathbb{R}^{n \times d}$, $f : \mathbb{R} \to \mathbb{R}$, suppose that $|f(x) - \sum_{j=0}^r c_j x^j| \leq \varepsilon$ in a closed interval containing all entries of $UV^\top$ for some $\varepsilon > 0$. Then, it holds that

$$\mathbb{E}\left[\left\|f^\circ(UV^\top) - \Gamma\right\|_F^2\right] \leq 2nr^2 \varepsilon^2 + \sum_{j=1}^r 2rc_j^2(2 + 3^j) \sum_k (\sum_i U_{ik}^2) \sum_i (\sum_k V_{ik}^2), \quad (3)$$

where $\Gamma$ is the output of Algorithm 2.

The proof of the above proposition is given in the supplementary material. Note that even when $\varepsilon$ is close to 0, the error bound (3) may increase exponentially with respect to the degree $r$. This is because the approximation error of TensorSketch grows exponentially with respect to the degree (see Theorem 1). Here, one can choose exponentially small (or zero) coefficient $c_j$ to compensate it, but it may hurt the approximation quality of $f$. Namely, it is highly non-trivial to balance two approximation components of Poly-TensorSketch: TensorSketch and a polynomial approximation for $f$. In the following section, we propose a novel approach to find the optimal coefficients minimizing the approximation error of Poly-TensorSketch.
3.2 Coefficient construction

A natural choice for the coefficients is to utilize a polynomial series such as Taylor, Chebyshev [14] or other orthogonal basis expansion [20]. However, constructing the coefficients in this way focuses on the error of the polynomial approximation of $f$, and ignores the error of TENSORSKETCH which depends on the decay of the coefficients, as reflected in the error bound (3). To address the issue, we study the following optimization to find optimal coefficients:

$$\min_{c \in \mathbb{R}^{r+1}} \mathbb{E} \left[ \| f^\odot (UV^\top) - \Gamma \|_F^2 \right],$$

where $\Gamma$ is the output of POLY-TENSORSKETCH and $c = [c_0, \ldots, c_r] \in \mathbb{R}^{r+1}$ is a vector of coefficients. However, it is not easy to solve the above optimization directly as its objective involves an expectation over random variables with huge support, i.e., uniform hash and binary sign functions. Instead, we aim for minimizing an upper bound of the approximation error (4). To this end, we define the following notation.

**Definition 1** Let $X \in \mathbb{R}^{n \times (r+1)}$ be the matrix $^3$ whose $k$-th column corresponds to the vectorization of $[UV^\top]_{ij}^{k-1}$ for all $i, j \in [n]$, $f \in \mathbb{R}^n$ by the vectorization of $f^\odot(UV^\top)$ and let $W \in \mathbb{R}^{(r+1) \times (r+1)}$ be a diagonal matrix such that $W_{ii} = 0$ and

$$W_{ii} = \sqrt{r(2 + 3^k)(\sum_j (\sum_k U_{jk}^2)^{\frac{1}{m}})(\sum_j (\sum_k V_{jk}^2)^{\frac{1}{m}})}, \quad \text{for } i = 2, \ldots, r + 1.$$

Using the above notation, we establish the following error bound.

**Lemma 1** Given $U, V \in \mathbb{R}^{n \times d}$ and $f : \mathbb{R} \rightarrow \mathbb{R}$, consider $X, f$ and $W$ defined in Definition 1. Then, it holds

$$\mathbb{E} \left[ \| f^\odot (UV^\top) - \Gamma \|_F^2 \right] \leq 2\| Xc - f \|_2^2 + 2\| Wc \|_2^2,$$

where $\Gamma$ is the output of Algorithm 2.

The proof of the above lemma is given in the supplementary material. Observe that the error bound (5) is a quadratic form of $c \in \mathbb{R}^{r+1}$, where it is straightforward to obtain a closed-form solution for minimizing it:

$$c^* := \arg \min_{c \in \mathbb{R}^{r+1}} \| Xc - f \|_2^2 + \| Wc \|_2^2 = \left( X^\top X + W^\top W \right)^{-1} X^\top f.$$  \hspace{1cm} (6)

This optimization task is also known as a generalized ridge regression [12]. The solution (6) minimizes the regression error (i.e., the error of polynomial), while it is regularized by $W$, i.e., $W_{ii}$ is a regularizer of $c_i$. Namely, if $W_{ii}$ grows exponentially with respect to $i$, then $c_i^*$ may decay exponentially (this compensates the error of TENSORSKETCH with degree $i$). By substituting $c^*$ into the error bound (5), we obtain the following multiplicative error bound of POLY-TENSORSKETCH.

**Theorem 2** Given $U, V \in \mathbb{R}^{n \times d}$ and $f : \mathbb{R} \rightarrow \mathbb{R}$, consider $X$ defined in Definition 1. Then, it holds

$$\mathbb{E} \left[ \| f^\odot (UV^\top) - \Gamma \|_F^2 \right] \leq \left( 1 + \frac{m\sigma^2}{rC} \right)^{-1} \| f^\odot(UV^\top) \|_F^2,$$

where $\Gamma$ is the output of Algorithm 2 with the coefficient $c^*$ in (6), $\sigma \geq 0$ is the smallest singular value of $X$ and $C = \max \left( \frac{5}{2} \right) \left( \frac{\| X \|_F^2 \| V \|_F}{\| f \|_F^2}, (2 + 3^r)(\sum_j (\sum_k U_{jk}^2)^{\frac{1}{m}})(\sum_j (\sum_k V_{jk}^2)^{\frac{1}{m}}) \right).$

The proof of the above theorem is given in the supplementary material. Observe that the error bound (7) is bounded by $\| f^\odot(UV^\top) \|_F^2$ since $m, r, \sigma, C \geq 0$. On the other hand, we recall that the error bound (3), i.e., POLY-TENSORSKETCH without using the optimal coefficient $c^*$, can grow exponentially with respect to $r$. We indeed observe that $c^*$ is empirically superior to coefficients of the popular Taylor and Chebyshev series expansions with respect to the error of POLY-TENSORSKETCH (see Section 4.1 for more details). We also remark that the error bound (7) of POLY-TENSORSKETCH is even better than that (1) of TENSORSKETCH even for the case of monomial $f(x) = x^k$. This is primarily because the former is achievable by paying an additional cost to compute the optimal coefficients (6). In what follows, we discuss the extra cost.

---

$^3$ $X$ is known as the Vandermonde matrix.
Sampling for efficient regression. To obtain the optimal coefficients $c^*$ in (6), one can check that $O(r^2 n^2 + r^3)$ operations are required because of computing $X^T X$ for $X \in \mathbb{R}^{n \times (r+1)}$ (see Definition 1). This hurts the overall complexity of POLY-TENSORSKETCH. Instead, we sample a few entries in $U^T V$ and approximately find the coefficients based on the samples. Formally, suppose each $(i, j) \in [n] \times [n]$ is chosen with probability $p_{ij}$ and $S$ be the set of the selections. Then, we use the following approximation:

$$c^* \approx \left( X^T \text{diag}(p)^{-1} X + W^T W \right)^{-1} X^T \text{diag}(p)^{-1} \tilde{f},$$

where $p := [p_{ij}] \in \mathbb{R}^{S}$ for $(i, j) \in S$, and $\tilde{f} \in \mathbb{R}^{S}$ (or $\tilde{X} \in \mathbb{R}^{S \times (r+1)}$) consists of values (or rows) in $f$ (or $X$) whose indices are in $S$. We suggest to sample $n$ entries at uniformly random, i.e., $|S| = n$. Then, computing $c^*$ in (8) requires $O(r^2 n + r^3) = O(n)$ for $r = O(1)$, and it marginally increases the overall complexity of POLY-TENSORSKETCH in our experiments. The choice (8) also shows comparable performance empirically with the exact one in (6).

Chebyshev polynomial regression for avoiding a numerical issue. Recall that $X$ contains $[U^T V]_{ij}^j$ (see Definition 1). If entries in $U^T V$ are greater (or smaller) than 1 and degree $r$ is large, $\tilde{X}$ can have huge (or negligible) values. This can cause a numerical issue for computing the optimal coefficients (6) (or (8)) using $X$. To alleviate the issue, we suggest to construct a matrix $X' \in \mathbb{R}^{n \times (r+1)}$ whose entries are the output of the Chebyshev polynomials: $[X']_{ij} = t_j([U^T V]_{ik})$ where $(k, ℓ) \in [n] \times [n]$ corresponds to index $i \in [n^2]$, $j \in [r + 1]$ and $t_j(x)$ is the Chebyshev polynomial of degree $j$ [14]. Now, the value of $t_j(x)$ is always in $[-1, 1]$ and does not monotonically increase or decrease with respect to the degree $j$. Then, we find the optimal coefficients $c' \in \mathbb{R}^{r+1}$ based on Chebyshev polynomials as follows:

$$c' = \left( X'^T X' + R^T W^T W R \right)^{-1} X'^T \tilde{f},$$

where $R \in \mathbb{R}^{(r+1) \times (r+1)}$ is defined to convert $c' \in \mathbb{R}^{r+1}$ into the coefficients based on monomials, i.e., $c^* = Rc'$ so that $\sum_{j=0}^{r} c_{j} x^j = \sum_{j=0}^{r} c_{j} t_j(x)$. We finally remark that to find $t_j$, one needs to find a closed interval containing all $[U^T V]_{ij}^j$. To this end, we use the interval $[-a, a]$ where $a = (\max_i \|U_{i,:}\|_2) (\max_j \|V_{j,:}\|_2)$ and $U_{i,:}$ is the $i$-row of the matrix $U$. It takes $O(nd)$ time and contributes marginally to the overall complexity of POLY-TENSORSKETCH.

4 Applications

In this section, we report the empirical results of POLY-TENSORSKETCH for the element-wise matrix functions. We first report its approximation error for the radial basis function (RBF) kernel matrices in Section 4.1. Then, we apply the approximation algorithm to two machine learning applications: classification and optimal transport, as reported in Section 4.2 and 4.3. All results are reported by averaging over 100 and 10 independent trials for the experiments in Section 4.1 and those in other sections, respectively. All real-world datasets used in this section are available at https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/.

4.1 RBF kernel approximation

We first benchmark our algorithm for approximating RBF kernels. Given $U = [u_1, \ldots, u_n]^T \in \mathbb{R}^{n \times d}$ of $n$ data samples, the RBF kernel $K = [K_{ij}]$ is defined as $K_{ij} := \exp(-\gamma \|u_i - u_j\|_2^2)$ for $i, j \in [n]$ and $\gamma > 0$. It can be represented using the element-wise matrix exponential function:

$$K = \text{diag}(-\gamma \exp(z)) \cdot \exp^\circ(2\gamma U U^T) \cdot \text{diag}(-\gamma \exp(z)) \in \mathbb{R}^{n \times n},$$

where $z = [\|u_1\|_2^2, \ldots, \|u_n\|_2^2]^T \in \mathbb{R}^n$. One can approximate the element-wise matrix function $\exp^\circ(2\gamma U U^T) \approx \Gamma$ where $\Gamma$ is the output of POLY-TENSORSKETCH with $f(x) = \exp(2\gamma x)$ so that $K \approx \text{diag}(-\gamma \exp(z)) \cdot \Gamma \cdot \text{diag}(-\gamma \exp(z))$. RBF kernel has been used in many applications including classification, [15], covariance estimation [22], Gaussian process [18], determinant point processes [1], where they commonly require multiplications between the kernel matrix and vectors.

For synthetic kernels, we generate random matrices $U \in \mathbb{R}^{1000 \times 50}$ whose entries are drawn from the normal distribution $\mathcal{N}(0, 1/50)$. For real-world kernels, we use segment and usps dataset. We report
We use the open source SVM package (LIBSVM) [4] and compare our algorithm with the kernel which is not the case for Taylor-TS and Chebyshev-TS (suboptimal versions of our algorithm). All test methods have comparable running times. One can expect that a linear SVM using a feature machine (SVM) based on RBF kernel. Given the input data \( U \in \mathbb{R}^{n \times d} \), our algorithm can find a feature \( T_U \in \mathbb{R}^{m \times m'} \) such that \( K \approx T_U T_U^T \) where \( K \) is the RBF kernel of \( U \) (see Section 4.1). One can expect that a linear SVM using \( T_U \) shows a similar performance compared to the kernel SVM using \( K \). However, for \( m' \ll n \), the complexity of a linear SVM is much cheaper than that of the kernel method both for training and testing. In order to utilize our algorithm, we construct

\[
T_U = \sqrt{c_0} T_U^{(0)} \cdots \sqrt{c_r} T_U^{(r)}
\]

where \( T_U^{(0)} \), \ldots, \( T_U^{(r)} \) are the \texttt{TensorSketch}s of \( U \) in Algorithm 2. Here, the coefficient \( c_j \) should be positive and one can compute the optimal coefficients (6) by adding non-negativity condition, which is solvable using simple quadratic programming with marginal additional cost.

We use the open source SVM package (LIBSVM) [4] and compare our algorithm with the kernel SVM using the exact RBF and the linear SVM using the embedded feature from RFF of the same running time with ours. We run all experiments with 10 cross-validations and report the average of the classification error on the validation dataset. We set \( m = 20 \), \( r = 3 \) for our algorithm and...
Table 1: Classification error, kernel approximation error and speedup for classification with kernel SVM under various real-world datasets.

| Dataset      | Statistics | Classification error (%) | Kernel approximation error | Speedup |
|--------------|------------|--------------------------|----------------------------|---------|
|              | n          | d           | Exact | RFF | Optimal-TS | RFF | Optimal-TS | Optimal-TS |
| segment      | 2310       | 19          | 3.51  | 3.75 | 3.46       | 9.76 × 10^{-3} | 6.24 × 10^{-4} | 7.27   |
| satimage     | 4335       | 36          | 23.02 | 23.8 | 22.83      | 3.7  | 5.46 × 10^{-2} | 17.77  |
| usps         | 7291       | 256         | 1.88  | 7.53 | 4.94       | 5.16 | 8.67 × 10^{-1} | 23.64  |
| phishing     | 11055      | 68          | 2.90  | 17.59| 8.58       | 8.88 × 10^{-3} | 1.61 × 10^{-4} | 51.25  |
| letter       | 15000      | 16          | 3.70  | 9.23 | 8.94       | 7.51 | 1.08        | 79.89  |

Figure 2: (a) Speedup per iteration (left) and the approximation ratio (right) of the Sinkhorn algorithm applied by our method and RFF. (b) Images tested in our experiments. The source image (left) is transported into the output (right) toward the target image (middle) using our method.

Given two images as shown in Figure 2 (b), we randomly sample \( \{x_i\}_{i=1}^n \) from RGB pixels in the source image and \( \{y_j\}_{j=1}^n \) from those in the target image. We set \( m = 20, d = 3, r = 3 \) and \( \gamma = 1 \). Figure 2 (a) reports the speedup per iteration of the tested approximation algorithms over the exact computation and their approximation ratios of the objective value in the Sinkhorn algorithm after 10 iterations. As reported in Figure 2 (a), both our algorithm and RFF run at orders of magnitude faster than the exact Sinkhorn algorithm. Furthermore, the approximation ratio of ours is much more stable, while RFF has a huge variance without any tendency on the dimension \( n \). In Figure 2 (b), we provide the transported image from the source image using the Sinkhorn algorithm utilizing our method.

5 Conclusion

In this paper, we design a fast algorithm for sketching element-wise matrix functions. Our method is to combine (a) a polynomial approximation with (b) the randomized matrix tensor sketch. Our main novelty is on finding the optimal polynomial coefficients for minimizing the overall approximation error bound where they balance the errors of (a) and (b). We demonstrate the applicability of our method in two applications and expect that the generic scheme would enjoy a broader usage in the future, e.g., for deep neural networks as we described in Section 1.
References

[1] Affandi, R. H., Fox, E., Adams, R., and Taskar, B. (2014). Learning the parameters of determinantal point process kernels. In International Conference on Machine Learning, pages 1224–1232.

[2] Altschuler, J., Weed, J., and Rigollet, P. (2017). Near-linear time approximation algorithms for optimal transport via sinkhorn iteration. In Advances in Neural Information Processing Systems, pages 1964–1974.

[3] Avron, H., Nguyen, H., and Woodruff, D. (2014). Subspace embeddings for the polynomial kernel. In Advances in Neural Information Processing Systems, pages 2258–2266.

[4] Chang, C.-C. and Lin, C.-J. (2011). LIBSVM: A library for support vector machines. ACM Transactions on Intelligent Systems and Technology, 2:27:1–27:27. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

[5] Charikar, M., Chen, K., and Farach-Colton, M. (2002). Finding frequent items in data streams. In International Colloquium on Automata, Languages, and Programming, pages 693–703. Springer.

[6] Cristianini, N., Shawe-Taylor, J., et al. (2000). An introduction to support vector machines and other kernel-based learning methods. Cambridge university press.

[7] Cuturi, M. (2013). Sinkhorn distances: Lightspeed computation of optimal transport. In Advances in neural information processing systems, pages 2292–2300.

[8] Flamary, R. and Courty, N. (2017). Pot python optimal transport library.

[9] Goodfellow, I., Bengio, Y., and Courville, A. (2016). Deep learning. MIT press.

[10] Han, I., Avron, H., and Shin, J. (2018). Stochastic chebyshev gradient descent for spectral optimization. In Advances in Neural Information Processing Systems, pages 7386–7396.

[11] Han, I., Malioutov, D., Avron, H., and Shin, J. (2016). Approximating the spectral sums of large-scale matrices using chebyshev approximations. arXiv preprint arXiv:1606.00942.

[12] Hemmerle, W. J. (1975). An explicit solution for generalized ridge regression. Technometrics, 17(3):309–314.

[13] Higham, N. J. (2008). Functions of matrices: theory and computation. SIAM.

[14] Mason, J. C. and Handscomb, D. C. (2002). Chebyshev polynomials. Chapman and Hall/CRC.

[15] Pennington, J., Yu, F. X. X., and Kumar, S. (2015). Spherical random features for polynomial kernels. In Advances in Neural Information Processing Systems, pages 1846–1854.

[16] Pham, N. and Pagh, R. (2013). Fast and scalable polynomial kernels via explicit feature maps. In Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining, pages 239–247. ACM.

[17] Rahimi, A. and Recht, B. (2008). Random features for large-scale kernel machines. In Advances in neural information processing systems, pages 1177–1184.

[18] Rasmussen, C. E. (2003). Gaussian processes in machine learning. In Summer School on Machine Learning, pages 63–71. Springer.

[19] Scholkopf, B. and Smola, A. J. (2001). Learning with kernels: support vector machines, regularization, optimization, and beyond. MIT press.

[20] Szeg, G. (1939). Orthogonal polynomials, volume 23. American Mathematical Soc.

[21] Ubaru, S., Chen, J., and Saad, Y. (2017). Fast estimation of tr(f(A)) via stochastic lanczos quadrature. SIAM Journal on Matrix Analysis and Applications, 38(4):1075–1099.
where we recall the Definition 1, that is,\( p_r(x) = \sum c_j x^T \) and \( [p_r^\circ(A)]_{ij} = p_r(A_{ij}) \). Consider the approximation error into the error from (1) polynomial approximation and (2) tensor sketch:

\[
E \left[ \|f^\circ(UV^T) - \Gamma\|^2_F \right] \leq E \left[ 2 \|f^\circ(UV^T) - p_r^\circ(UV^T)\|^2_F + 2 \|p_r^\circ(UV^T) - \Gamma\|^2_F \right]
\]

\[
= 2 \|f^\circ(UV^T) - p_r^\circ(UV^T)\|^2_F + 2E \left[ \|p_r^\circ(UV^T) - \Gamma\|^2_F \right]
\]

where the inequality comes from that \((a + b)^2 \leq 2(a^2 + b^2)\) for \(a, b \in \mathbb{R}\). The first error is straightforward from the assumption:

\[
\|f^\circ(UV^T) - \Gamma\|^2_F = \sum_{i,j} (f((UV^T)_{ij}) - p_r((UV^T)_{ij}))^2 \leq n^2 \varepsilon^2
\]

For the second error, we use Theorem 1 to have

\[
E \left[ \|p_r^\circ(UV^T) - \Gamma\|^2_F \right] \leq r \sum_{j=1}^r c_j^2 E \left[ \left\| (UV^T)^{\otimes j} - T_U^{(j)} T_V^{(j)^T} \right\|^2_F \right]
\]

\[
\leq r \sum_{j=1}^r c_j^2 (2 + 3^j) \left( \sum_i (\sum_k U_{ik}^2)^j \right) \left( \sum_i (\sum_k V_{ik}^2)^j \right)
\]

Putting all together, we conclude the result and this completes the proof of Proposition 1. \(\blacksquare\)

### A.2 Proof of Lemma 1

**Proof.** We recall that \( \Gamma = \sum_{j=0}^r c_j T^{(j)} \), and similar to the proof of Proposition 1 we have

\[
E \left[ \|f^\circ(UV^T) - \Gamma\|^2_F \right] \leq 2 \|f^\circ(UV^T) - p_r^\circ(UV^T)\|^2_F + 2E \left[ \|p_r^\circ(UV^T) - \Gamma\|^2_F \right]
\]

where the inequality comes from that \((a + b)^2 \leq 2(a^2 + b^2)\). The error in the first term can be written as

\[
\|f^\circ(UV^T) - p_r^\circ(UV^T)\|^2_F = \sum_{i,j} (f((UV^T)_{ij}) - p_r((UV^T)_{ij}))^2 = \|X \mathbf{c} - \mathbf{f}\|^2_2
\]

where we recall the Definition 1, that is,

\[
X := \begin{bmatrix}
1 & [UV^T]_{11} & [UV^T]_{12}^2 & \ldots & [UV^T]_{12}^r \\
1 & [UV^T]_{12} & [UV^T]_{12}^2 & \ldots & [UV^T]_{12}^r \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
1 & [UV^T]_{nn} & [UV^T]_{nn}^2 & \ldots & [UV^T]_{nn}^r
\end{bmatrix} \in \mathbb{R}^{n^2 \times (1 + r)}
\]
(i.e., also known as the Vandermonde matrix) and \( f \in \mathbb{R}^{n^2} \) is the vectorization \( f \left((UV^T)_{ij}\right) \). For the second error, we use Theorem 1 to have

\[
E \left[ \| p^w_f (UV^T) - \Gamma \|_F^2 \right] \leq r \sum_{j=1}^{r} c_j^2 E \left[ \| (UV^T)^{\otimes j} - T_j^{(i)} T_j^{(i)\top} \|_F^2 \right]
\]

\[
\leq r \sum_{j=1}^{r} c_j^2 (2 + 3k) \left( \sum_i (\sum_k U_{ik})^j \right) \left( \sum_i (\sum_k V_{ik})^j \right)
\]

\[
= \| Wc \|_2^2
\]

where \( W \in \mathbb{R}^{(r+1) \times (r+1)} \) is defined as a diagonal matrix with (see Definition 1)

\[
W_{ii} = \begin{cases} 
\frac{r(2 + 3k)}{m} \left( \sum_j (\sum_k U_{jk})^i \right) \left( \sum_j (\sum_k V_{jk})^i \right) & \text{if } i = 2, \ldots, r+1 \\
0, & \text{if } i = 1.
\end{cases}
\]

Putting all together, we have the results, that is,

\[
E \left[ \| f^o (UV^T) - \Gamma \|_F^2 \right] \leq 2 \left( \| Xc - f \|_2^2 + \| Wc \|_2^2 \right).
\]

This completes the proof of Lemma 1.

\section{A.3 Proof of Theorem 2}

\textbf{Proof.} We denote that

\[
g(c) := \| Xc - f \|_2^2 + \| Wc \|_2^2 = c^\top (X^\top X + W^\top W)c - 2f^\top Xc + f^\top f
\]

for \( c \in \mathbb{R}^{r+1} \) and substituting \( c^* = (X^\top X + W^\top W)^{-1} X^\top f \) into the above, we have

\[
g(c^*) = f^\top (I - X(X^\top X + W^\top W)^{-1} X^\top) f
\]

\[
\leq \| I - X(X^\top X + W^\top W)^{-1} X^\top \|_2 \| f \|_2^2
\]

\[
= \| I - X(X^\top X + W^\top W)^{-1} X^\top \|_2 \| f^o(UV^T) \|_F^2
\]

(9)

Since \( X^\top X \) is positive semi-definite, it has eigendecomposition as \( X^\top X = V \Sigma V^\top \) and \( X = \Sigma^{1/2} V^\top \). Then, we have

\[
X(X^\top X + W^\top W)^{-1} X^\top = \Sigma^{1/2} V^\top (V \Sigma V^\top + W^\top W)^{-1} V^\top \Sigma^{1/2}
\]

\[
= \left( I + \Sigma^{-1/2} V^\top W^\top W V \Sigma^{-1/2} \right)^{-1}
\]

For simplicity, let \( M := \Sigma^{-1/2} V^\top W^\top W V \Sigma^{-1/2} \) and observe that \( M \) is symmetric and positive semi-definite because \( W \) is a diagonal and \( W_{ii} \geq 0 \) for all \( i \) (see Definition 1). And the upper bound (9) can be written as

\[
g(c^*) \leq \| I - (I + M)^{-1} \|_2 \| f^o(UV^T) \|_F^2.
\]

Then, we have

\[
\| I - (I + M)^{-1} \|_2 = \| M(I + M)^{-1} \|_2 = \frac{\| M \|_2}{1 + \| M \|_2}
\]

(10)

since \( x/(1 + x) \) is a increasing function. From the submultiplicativity of \( \| \cdot \|_2 \), we have

\[
\| M \|_2 = \left\| \Sigma^{-1/2} V^\top W^\top W V \Sigma^{-1/2} \right\|_2
\]

\[
\leq \left\| \Sigma^{-1/2} \right\|_2 \left\| V^\top \right\|_2 \left\| W^\top W \right\|_2 \left\| V \right\|_2 \left\| \Sigma^{-1/2} \right\|_2
\]

\[
= \left\| \Sigma^{-1} \right\|_2 \left\| W^\top W \right\|_2.
\]

(11)
where the last equality is from \( \|V\|_2 = 1 \). We remind that
\[
W_{ii} = \sqrt{\frac{r(2 + 3^i)(\sum_j (\sum_k U_{jk}^2)^i)(\sum_j (\sum_k V_{jk}^2)^i)}{m}}
\]
for \( i \in 2, \ldots, r + 1 \) and \( W_{1,1} = 0 \). Therefore,
\[
\|W^T W\|_2 = \max_i W_{ii}^2
\]
\[
= \frac{r}{m} \max \left( 5\|U\|_F^2\|V\|_F^2 (2 + 3^r) \left( \sum_j \left( \sum_k U_{jk}^2 \right)^r \right) \left( \sum_j \left( \sum_k V_{jk}^2 \right)^r \right) \right)
\]
\[
= \frac{rC}{m}
\]
and recall that \( \sigma \geq 0 \) is denoted by the smallest singular value of \( X \). Then, \( \|\Sigma^{-1}\|_2 \leq 1/\sigma^2 \).
Substituting the above bounds on \( \|W^T W\|_2, \|\Sigma^{-1}\|_2 \) into (11), we have
\[
\|M\|_2 \leq \frac{rC}{m\sigma^2}
\]
and putting this bound and (10) into (9), we have that
\[
g(c^*) \leq \left( 1 + \frac{m\sigma^2}{rC} \right)^{-1} \|f\|_2^2 = \left( 1 + \frac{m\sigma^2}{rC} \right)^{-1} \|f^\circ(UV^T)\|_F^2.
\]
This completes the proof of Theorem 2. \( \blacksquare \)