Near Input Sparsity Time Kernel Embeddings via Adaptive Sampling

David P. Woodruff [*1] **Amir Zandieh** [*2]

Abstract

To accelerate kernel methods, we propose a near input sparsity time algorithm for sampling the high-dimensional feature space implicitly defined by a kernel transformation. Our main contribution is an importance sampling method for subsampling the feature space of a degree q tensoring of data points in almost input sparsity time, improving the recent oblivious sketching method of (Ahle et al., 2020) by a factor of $q^{5/2}/\epsilon^2$. This leads to a subspace embedding for the polynomial kernel, as well as the Gaussian kernel, with a target dimension that is only linearly dependent on the statistical dimension of the kernel and in time which is only linearly dependent on the sparsity of the input dataset. We show how our subspace embedding bounds imply new statistical guarantees for kernel ridge regression. Furthermore, we empirically show that in large-scale regression tasks, our algorithm outperforms state-of-the-art kernel approximation methods.

1. Introduction

Kernel methods provide a simple, yet powerful framework for applying non-parametric modeling techniques to a number of important problems in statistics and machine learning, such as kernel ridge regression, SVM, PCA, CCA, etc. While kernel methods are statistically well understood and perform well empirically, they often pose scalability challenges as they operate on the kernel matrix (Gram matrix) of the data, whose size scales quadratically in the size of the training dataset. Primitives such as kernel PCA or kernel ridge regression generally take a prohibitively large quadratic amount of space and at least quadratic time. Thus, much work has focused on scaling up kernel methods by producing compressed and *low-rank* approximations to kernel matrices (Rahimi & Recht, 2008; Alaoui & Mahoney, 2015; Avron et al., 2017a; Musco & Musco, 2017; Avron et al., 2017b; 2014; Ahle et al., 2020; Zandieh et al., 2020).

[*]Equal contribution [1]Carnegie Mellon University, USA [2]Ecole polytechnique federale de Lausanne, Switzerland. Part of this work was done while the author was visiting CMU. Correspondence to: David P. Woodruff <dwoodruf@cs.cmu.edu>, Amir Zandieh <amir.zandieh@epfl.ch>.

Proceedings of the 37^{th} International Conference on Machine Learning, Online, PMLR 119, 2020.

1.1. Problem Definition

For a given kernel function $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ and a dataset of d-dimensional vectors $x_1, x_2, \cdots x_n \in \mathbb{R}^d$, let $K \in \mathbb{R}^{n\times n}$ be the kernel matrix corresponding to this dataset defined as $K_{ij} = k(x_i, x_j)$ for every $i, j \in [n]$. A classical solution for scaling up kernel methods is via kernel *low-rank approximation*, where one seeks to find a low-rank matrix $Z \in \mathbb{R}^{s\times n}$ such that $Z^\top Z$ can serve as a proxy to the kernel matrix K. In order to obtain statistical and algorithmic guarantees for downstream kernel-based learning applications, such as kernel regression, PCR, CCA, etc., one needs to have spectral approximation bounds on the entire surrogate kernel matrix. Formally, for given $\epsilon, \lambda > 0$ we need $Z^\top Z$ to be an (ϵ, λ)-*spectral approximation* to the kernel matrix K, meaning that $Z^\top Z$ has to satisfy,

$$\frac{K + \lambda I}{1+\epsilon} \preceq Z^\top Z + \lambda I \preceq \frac{K + \lambda I}{1-\epsilon}. \tag{1}$$

Intuitively, if λ is much larger than the operator norm of K then $Z = 0$ is a good solution that satisfies (1). On the other hand if $\lambda = 0$, then the target dimension s has to be at least equal to the rank of K. In general, the *statistical dimension* (or *effective dimension*) captures this tradeoff, defined as $s_\lambda := \sum_{i=1}^n \frac{\lambda_i}{\lambda_i+\lambda}$, where the λ_i are the eigenvalues of K. The goal is to find a matrix $Z \in \mathbb{R}^{s\times n}$ with a target dimension s which depends only linearly on s_λ, using a runtime that is nearly equal to the number of non-zero entries (i.e., the sparsity) of the input dataset, denoted by $\text{nnz}(X)$. The main motivation of this paper is the following:

P : *Given a dataset $x_1, x_2, \cdots x_n \in \mathbb{R}^d$, and a kernel function $k(\cdot)$, if K is the kernel matrix corresponding to this dataset with statistical dimension $s_\lambda = \mathbf{tr}\left(K(K+\lambda I)^{-1}\right)$, can we compute a matrix $Z \in \mathbb{R}^{s\times n}$ with $s = O\left(\frac{s_\lambda}{\epsilon^2}\log n\right)$, using $O\left(\textit{poly}(s_\lambda, \frac{1}{\epsilon}, \log n)\cdot n + \textit{poly}(\log n)\cdot \textit{nnz}(X)\right)$ runtime, such that $Z^\top Z$ is an (ϵ, λ)-spectral approximation to K as per (1)?*

The runtime that **(P)** is asking for requires the $\text{poly}(s_\lambda, \epsilon^{-1})$

terms to be decoupled from the input sparsity, $\text{nnz}(X)$. Hence, up to low order terms, we aim for a runtime which only depends linearly on the sparsity of the input dataset.

We address **(P)** for two important kernel classes: the degree-q polynomial kernel $k(x,y) = \langle x, y\rangle^q$ for some $q \in \mathbb{Z}_+$, and the Gaussian kernel $k(x,y) = e^{-\|x-y\|_2^2/2}$. We also remark that, as we will later discuss in Section 3.3, our method is very general and can be applied to the class of dot-product kernels. As we will discuss in the related work section, all prior methods for approximating the polynomial kernel achieve a runtime of either the form $\text{poly}(\epsilon^{-1}, q, \log n) \cdot \text{nnz}(X)$ or $\text{poly}(s_\lambda, \epsilon^{-1}, \log n) \cdot \text{nnz}(X)$, and similarly all prior results for the Gaussian kernel achieve a runtime of either $\text{poly}(\epsilon^{-1}, r, \log n) \cdot \text{nnz}(X)$ or $\text{poly}(s_\lambda, \epsilon^{-1}, \log n) \cdot \text{nnz}(X)$, where r is the radius of the input dataset. These are strictly worse than the target runtime of **(P)**.

1.2. Our Results

We answer problem **(P)** in the affirmative by designing near input sparsity time algorithms for embedding the polynomial and Gaussian kernels. Our main result for the the polynomial kernel is given in the following theorem.

Theorem 1. *For any dataset $x_1, \cdots x_n \in \mathbb{R}^d$, any $\epsilon, \lambda > 0$ and any positive integer q, if $K \in \mathbb{R}^{n\times n}$ is the degree-q polynomial kernel matrix corresponding to this dataset ($K_{i,j} := \langle x_i, x_j\rangle^q$) with statistical dimension s_λ and $\frac{\mathbf{tr}(K)}{\epsilon\lambda} = O(poly(n))$, then there exists an algorithm that computes a matrix $Z \in \mathbb{R}^{s\times n}$, with target dimension $s = O\left(\frac{s_\lambda}{\epsilon^2}\log n\right)$ such that, with high probability, $Z^\top Z$ is an (ϵ, λ)-spectral approximation to K as in (1) using $O\left(poly(\epsilon^{-1}, q, \log n)\cdot s_\lambda^2 n + q^{5/2}\log^4 n \cdot nnz(X)\right)$ time.*

We also address $(\mathbf{P})$ for approximating the Gaussian kernel by proving the following theorem.

Theorem 2. *For any dataset $x_1, \cdots x_n \in \mathbb{R}^d$ such that $\|x_i\|_2^2 \le r$ for all $i \in [n]$, any $\epsilon, \lambda \ge \frac{1}{poly(n)}$, if $K \in \mathbb{R}^{n\times n}$ is the Gaussian kernel matrix corresponding to this dataset ($K_{i,j} := e^{-\|x_i - x_j\|_2^2/2}$) with statistical dimension s_λ, then there exists an algorithm that computes a matrix $Z \in \mathbb{R}^{s\times n}$, with target dimension $s = O\left(\frac{s_\lambda}{\epsilon^2}\log n\right)$ such that, with high probability, $Z^\top Z$ is an (ϵ, λ)-spectral approximation to K as in (1) using $O\left(poly(\epsilon^{-1}, r, \log n)\cdot s_\lambda^2 n + r^{5/2}\log^4 n \cdot nnz(X)\right)$ time.*

Theorems 1 and 2 imply accelerated algorithms for kernel ridge regression (KRR) with improved statistical and algorithmic guarantees. We analyze the empirical risk of our sampling algorithm for the KRR problem in Appendix H. Furthermore, in the experiments section we evaluate our approximate KRR method on various standard large-scale regression datasets and empirically show that our method competes favorably with the state-of-the-art, including Nystrom (Musco & Musco, 2017) and Fourier features methods (Rahimi & Recht, 2008), as well as the oblivious sketching of (Ahle et al., 2020). We show that our method achieves better testing error and smaller runtime on large datasets with more than half a million training examples.

Additional downstream learning applications: While we focus on KRR here, we remark that spectral approximation bounds form the basis of analyzing sketching methods for tasks including kernel low-rank approximation, PCA, CCA, k-means and many more. In the kernelized setting, such bounds have been analyzed, without regularization, for the polynomial kernel (Avron et al., 2014). It is shown in (Cohen et al., 2017) that (1) along with a trace condition on $Z^\top Z$ (which holds for the sampling approaches we consider) yields a so-called *projection-cost preservation* condition. With λ chosen appropriately, this condition ensures that $Z^\top Z$ can serve as a proxy for K for approximately solving kernel k-means and for certain versions of kernel PCA and kernel CCA. See (Musco & Musco, 2017) for details, where this analysis is carried out for the Nystrom method.

1.3. Prior Work

A popular approach for accelerating kernel methods is based on Nystrom sampling. We refer the reader to the work of (Musco & Musco, 2017) and the references therein. By recursively sampling Nystrom landmarks according to the so-called ridge leverage score distribution, Musco & Musco (2017) prove that for any kernel K with statistical dimension s_λ, there exists an algorithm that outputs a matrix $Z \in \mathbb{R}^{s\times n}$ with $s = O\left(\frac{s_\lambda}{\epsilon}\log n\right)$ which satisfies the spectral approximation guarantee of (1) with high probability, using $O\left(n\frac{s_\lambda^2}{\epsilon^2}\cdot\log^2 n + \frac{s_\lambda}{\epsilon}\log n\cdot\text{nnz}(X)\right)$ runtime. However, the leading term in the time complexity of this method is $O\left(\frac{s_\lambda}{\epsilon}\log n\cdot\text{nnz}(X)\right)$, which unsatisfactorily depends on ϵ^{-1} and also depends linearly on s_λ. Hence, for both the polynomial and Gaussian kernels our Theorems 1 and 2 improve on the runtime of this method by a factor of $\epsilon^{-1}s_\lambda$.

Another popular line of work on kernel approximation problems is the Fourier features method of Rahimi & Recht (2008). It is proved in (Avron et al., 2017b) that this method can achieve spectral approximation guarantees for the Gaussian kernel using a sub-optimal number $s \approx \epsilon^{-2}\frac{n}{\lambda}\log n$ of samples and $O\left(\epsilon^{-2}\frac{n}{\lambda}\log n\cdot\text{nnz}(X)\right)$ runtime. This sample complexity is substantially larger than our result in Theorem 2. Furthermore we improve the runtime of this method by a factor of $\epsilon^{-2}\frac{n}{\lambda}$. However, (Avron et al., 2017b) show that this method can be modified to achieve a sample complexity of $s = \Theta(1)^d\cdot\frac{s_\lambda}{\epsilon^2}\log n$ using a runtime of $\Theta(1)^d\cdot\frac{s_\lambda}{\epsilon^2}\log n\cdot\text{nnz}(X)$. For constant dimensional datasets (constant d) the number of samples that (Avron et al., 2017b) achieve is comparable to our target dimension in Theorem 2

but it deteriorates exponentially with the dimension d. Furthermore, the runtime of this method is substantially larger than our runtime by a factor of $\Theta(1)^d \cdot \epsilon^{-2} s_\lambda$.

In the linear sketching literature, (Avron et al., 2014) proposed an *oblivious subspace embedding* for the polynomial kernel based on the TensorSketch of (Pham & Pagh, 2013). They applied this method to a wide array of kernel problems, including PCA, PCR, CCA. The runtime of this method, while nearly linear in $\text{nnz}(X)$, scales *exponentially* in the degree q of the polynomial kernel. Their runtime for the degree-q polynomial kernel is $O\left(\frac{q \cdot 3^q s_\lambda^2}{\epsilon^2} + q \cdot \text{nnz}(X)\right)$, which has an unsatisfactory 3^q term.

Recently, (Ahle et al., 2020) proposed a new *oblivious sketching* solution for the polynomial kernel that improves the exponential dependence of TensorSketch on q to polynomial. Ahle et al. (2020) gave an algorithm that outputs a matrix $Z \in \mathbb{R}^{s \times n}$ with $s = \widetilde{O}\left(\frac{q^4 s_\lambda}{\epsilon^2}\right)$ which satisfies the spectral approximation guarantee of (1) with high probability. Their algorithm has $\widetilde{O}\left(\frac{q^5 s_\lambda}{\epsilon^2} \cdot n + \frac{q^5}{\epsilon^2} \cdot \text{nnz}(X)\right)$ runtime[1]. This runtime has an undesirable inverse polynomial dependence on ϵ and scales sub-optimally with the degree of the polynomial kernel as q^5. Our Theorem 1 improves the runtime of (Ahle et al., 2020) by an $\epsilon^{-2} q^{5/2}$ factor. Moreover, they showed that their sketch for the polynomial kernel leads to an efficient oblivious sketch for the Gaussian kernel on bounded datasets. Ahle et al. (2020) gave an algorithm that for any dataset $x_1, x_2, \cdots x_n \in \mathbb{R}^d$ with radius r, computes a matrix $Z \in \mathbb{R}^{s \times n}$ with $s = \widetilde{O}\left(\frac{r^5 s_\lambda}{\epsilon^2}\right)$ which spectrally approximates the Gaussian kernel matrix corresponding to this dataset as in (1) with high probability. This was the first result that resolved the curse of dimensionality for embedding the high dimensional Gaussian kernel. The algorithm has $\widetilde{O}\left(\frac{r^6 s_\lambda}{\epsilon^2} \cdot n + \frac{r^6}{\epsilon^2} \cdot \text{nnz}(X)\right)$ runtime, which unsatisfactorily depends on $1/\epsilon^2$ and scales poorly as a function of the dataset's radius as r^6. Our Theorem 2 improves this runtime by a factor of $\epsilon^{-2} r^{7/2}$.

1.4. Our Techniques

Our method relies on the fact that any kernel function $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ defines a lifting ϕ such that the kernel function computes the inner product between the lifted data points, i.e., $k(x, y) = \langle \phi(x), \phi(y) \rangle$. Therefore, any kernel matrix K can be decomposed as $K = \Phi^\top \Phi$ where Φ is a matrix with n columns whose columns are the lifted data points $\phi(x_i)$. Our approach is to design an importance sampling matrix Π such that $Z = \Pi\Phi$ satisfies the spectral approximation guarantee of (1). Our algorithm generates a sampling matrix Π that samples a small number of rows of Φ using a *recursive leverage score* sampling technique, which has been extensively applied to various algorithmic problems in the literature (Kapralov et al., 2014; Alaoui & Mahoney, 2015; Cohen et al., 2016; Musco & Musco, 2017; Avron et al., 2017b; Cohen et al., 2017). Our main novelty is in generating a sample from the leverage score distribution without ever forming the entire distribution explicitly, as the support size of this distribution is equal to the number of rows of Φ which is typically high (even infinite).

For the polynomial kernel, the lifting matrix is $\Phi = X^{\otimes q}$, where $X^{\otimes q}$ is a $d^q \times n$ matrix whose columns are obtained by a q-fold self-tensoring of the columns of the dataset matrix $X \in \mathbb{R}^{d \times n}$ (see Section 2 for notation). After multiple reductions, our importance sampling problem boils down to performing ℓ_2-sampling on a vector of the form $X^{\otimes q} v$, where v is an arbitrary vector in $\mathbb{R}^n$. Here by ℓ_2-sampling of a vector, we mean sampling a coordinate proportional to its squared value. We design a primitive that can generate a sample $i \in [d^q]$ with probability proportional to the squared value of the i^{th} entry of the vector $X^{\otimes q} v$ using roughly $\text{nnz}(X)$ time. Our algorithm relies on the fact that, by reshaping, entries of the vector $X^{\otimes q} v$ are in bijective correspondence with entries of the matrix $X^{\otimes q-1} \cdot \text{diag}(v) X^\top$, where $\text{diag}(v)$ is a diagonal $n \times n$ matrix whose diagonal entries are the elements of v. Therefore, our importance sampling amounts to sampling an element of $X^{\otimes q-1} \cdot \text{diag}(v) X^\top$ with probability proportional to the square of its absolute value. We do this by first sampling a column of this matrix with probability proportional to its squared norm, and then sampling a row with probability proportional to the squares of the entries of the sampled column. After sampling a column $l \in [d]$ of the matrix $X^{\otimes q-1} \cdot \text{diag}(v) X^\top$, we next perform ℓ_2-sampling on the l^{th} column of the mentioned matrix, which is in the form of $X^{\otimes q-1} u$, where $u = \text{diag}(v) X^\top_{l,\star}$. One can see that we have made progress and now it is enough to iterate in this fashion by performing ℓ_2-sampling on $X^{\otimes q-1} u$. However, note that $X^{\otimes q-1} \cdot \text{diag}(v) X^\top$ has d^{q-1} rows, and hence, computing its column norms is prohibitively expensive. We tackle this issue by sketching the columns of $X^{\otimes q-1} \cdot \text{diag}(v) X^\top$ using the sketch introduced in (Ahle et al., 2020), which is able to preserve the column norms up to a small error and with runtime roughly $\text{nnz}(X)$.

Our algorithm is actually more involved and includes extra dimensionality reduction steps. In the paragraph above we explained how to generate a single sample with the right distribution, but in order to obtain the spectral approximation guarantee of (1) we need to generate $s = O\left(\frac{s_\lambda}{\epsilon^2} \log n\right)$ such samples. It is crucial that our runtime does not lose a multiplicative factor of s. We heavily exploit the structure of tensor products to reuse most computations and generate s samples in time proportional to $\text{nnz}(X)$. Moreover, to spectrally approximate the Gaussian kernel, we adapt

[1] $\widetilde{O}$ notation hides $\text{poly}(\log n)$ factors.

Algorithm 1 RECURSIVE LEVERAGE SCORE SAMPLING

input: Matrix $\Phi \in \mathbb{R}^{D\times n}$, $\lambda \in \mathbb{R}_+$, $\epsilon \in \mathbb{R}_+$, $\mu \in \mathbb{R}_+$
output: Sampling matrix $\Pi \in \mathbb{R}^{s\times D}$

$s \leftarrow C\frac{\mu}{\epsilon^2}\log_2 n$ for some constant C
$S_0 \leftarrow \{0\}^{1\times D}$
$\lambda_0 \leftarrow \frac{\|\Phi\|_F^2}{\epsilon}$
$T \leftarrow \lceil \log_2 \frac{\lambda_0}{\lambda} \rceil$
for $t = 1$ to T **do**
  $S_t \leftarrow$ ROWSAMPLER $(\Phi, S_{t-1}\Phi, \lambda_{t-1}, s)$
  $\lambda_t = \lambda_{t-1}/2$
end for
return $\Pi = S_T$

our sampling algorithm to a truncated Taylor expansion of the Gaussian kernel. Furthermore, in Section 3.3 we discuss how our method can be generalized to any dot-product kernel.

2. Preliminaries

Throughout the paper, for any matrices $A \in \mathbb{R}^{m\times n}$ and $B \in \mathbb{R}^{d\times n}$, $A \oplus B \in \mathbb{R}^{(m+d)\times n}$ denotes the vertical concatenation of A and B, i.e., $A \oplus B = \begin{bmatrix} A \\ B \end{bmatrix}$.

Moreover, $A \otimes B \in \mathbb{R}^{(md)\times n}$ denotes the vertical tensor product of A and B. The rows of $A \otimes B$ are indexed by (i,j) where $i \in [m]$ and $j \in [d]$ and for any $l \in [n]$, $[A \otimes B]_{(i,j),l} = A_{i,l} \cdot B_{j,l}$. We also use $A^{\otimes q}$ to denote, $A^{\otimes q} = \underbrace{A \otimes A \cdots \otimes A}_{q \text{ terms}}$.

For any matrix X we use $X_{i,\star}$ to denote its i^{th} row and we use $X_{\star,i}$ to refer to its i^{th} column. Also for any set S, $X_{S,\star}$ denotes a sub-matrix of X that includes rows $i \in S$ of X.

3. Algorithm and Analysis

Let $\Phi \in \mathbb{R}^{D\times n}$ be the feature matrix whose columns are the projections of the data points in the feature space. We start by presenting a recursive importance sampling algorithm that efficiently computes a matrix Z which satisfies the spectral approximation guarantee of (1) for the kernel $K = \Phi^\top\Phi$. Sampling rows of Φ with probabilities proportional to the squared row norms of the matrix $\Phi(\Phi^\top\Phi + \lambda I)^{-1/2}$, which are known as the *ridge leverage scores* of Φ, is an efficient sampling strategy for obtaining the spectral approximation guarantee of (1). In Algorithm 1, we give a generic recursive method for performing approximate leverage score sampling on any matrix Φ. The recursive procedure works by generating samples from a crude approximation to the leverage scores and iteratively refining the sampling distribution. We first introduce the definition of a *row norm sampler* as follows,

Definition 3.1 (Row Norm Sampler)**.** Let Φ be a $D \times n$ matrix with rows $\phi_1, \phi_2, \cdots \phi_D \in \mathbb{R}^n$. For any probability distribution $\{p_i\}_{i=1}^D$ that satisfies $p_i \geq \frac{1}{4}\frac{\|\phi_i\|_2^2}{\|\Phi\|_F^2}$ for all $i \in [D]$, and any positive integer s, a *rank-s row norm sampler* for matrix Φ is a random matrix $S \in \mathbb{R}^{s\times D}$ which is constructed by generating s i.i.d. samples $j_1, j_2, \cdots j_s \in [D]$ with distribution $\{p_i\}_{i=1}^D$ and letting the r^{th} row of S be $\frac{1}{\sqrt{sp_{j_r}}}\mathbf{e}_{j_r}^\top$ for every $r \in [s]$, where $\mathbf{e}_1, \mathbf{e}_2, \cdots \mathbf{e}_D \in \mathbb{R}^D$ are the standard basis vectors in $\mathbb{R}^D$.

Now we are ready to prove the correctness of Algorithm 1,

Lemma 3. *Suppose that for any matrices $\Phi \in \mathbb{R}^{D\times n}$ and $B \in \mathbb{R}^{m\times n}$, any $\lambda' > 0$, and any positive integer s', the primitive* ROWSAMPLER(Φ, B, λ', s') *returns a rank-s' row norm sampler for matrix $\Phi(B^\top B + \lambda' I)^{-1/2}$ as in Definition 3.1. Then for any matrix $\Phi \in \mathbb{R}^{D\times n}$ with statistical dimension $s_\lambda = \|\Phi(\Phi^\top\Phi + \lambda I)^{-1/2}\|_F^2$, any $\lambda, \epsilon > 0$, any $\mu \geq s_\lambda$, Algorithm 1 returns a sampling matrix $\Pi \in \mathbb{R}^{s\times d}$ with $s = O(\frac{\mu}{\epsilon^2}\log n)$ such that with probability $1 - \frac{1}{poly(n)}$,*

$$\frac{\Phi^\top\Phi + \lambda I}{1+\epsilon} \preceq \Phi^\top\Pi^\top\Pi\Phi + \lambda I \preceq \frac{\Phi^\top\Phi + \lambda I}{1-\epsilon}.$$

The proof of this lemma is included in Appendix C.

3.1. Adaptive Sampling for the Polynomial Kernel

The polynomial kernel of degree q is defined as $k(x,y) = \langle x, y\rangle^q$. Using the definition of tensor products, one can see that $\langle x, y\rangle^q = \langle x^{\otimes q}, y^{\otimes q}\rangle$, where $x^{\otimes q}$ and $y^{\otimes q}$ are q-fold self tensor products of vectors x and y, respectively. Suppose $X \in \mathbb{R}^{d\times n}$ is the dataset matrix. The polynomial kernel matrix can be decomposed as $K = (X^{\otimes q})^\top X^{\otimes q}$, where $X^{\otimes q}$ is a $d^q \times n$ matrix whose columns are obtained by the q-fold self tensoring of the columns of X. The goal is to apply the iterative leverage score sampling of Algorithm 1 to the feature matrix $\Phi = X^{\otimes q}$ in nearly nnz(X) time. Note that the matrix Φ has a large number d^q of rows so even assuming that an oracle gives us the leverage score distribution of Φ for free, just reading this distribution takes d^q time. We show how to generate samples from the right distribution quickly.

Algorithm 1 crucially uses the primitive ROWSAMPLER, which carries out the main computations of our proposed algorithm. This primitive performs row norm sampling (see Definition 3.1) on a matrix of the form $\Phi(B^\top B + \lambda I)^{-1/2}$, for any matrix B, very efficiently.

3.1.1. ROWSAMPLER FOR THE POLYNOMIAL KERNEL

An important technical contribution of this work is an efficient algorithm that can perform row norm sampling (see Definition 3.1) on a matrix of the form $X^{\otimes q}(B^\top B + \lambda I)^{-1/2}$ using nearly nnz(X) runtime, where $X \in \mathbb{R}^{d\times n}$

and $B \in \mathbb{R}^{m\times n}$. Our primitive uses the sketch which was proposed in (Ahle et al., 2020) to preserve the norm of vectors in $\mathbb{R}^{d^q}$ and sketch vectors of the form $x^{\otimes q}$ quickly. The next lemma follows from Theorem 1.2 of (Ahle et al., 2020),

Lemma 4. *For every positive integers q, d, every $\epsilon > 0$, and every $\delta > 0$, there exists a distribution on random matrices $Q^q \in \mathbb{R}^{m\times d^q}$ with $m = O\left(\frac{1}{\epsilon^2}\log\frac{1}{\delta}\right)$ such that, $\Pr\left[\|Q^q y\|_2^2 \in (1\pm\epsilon)\|y\|_2^2\right] \geq 1-\delta$ for any $y \in \mathbb{R}^{d^q}$. Moreover, for any $x \in \mathbb{R}^d$, the total time to compute $Q^q\left(x^{\otimes q-j}\otimes \mathbf{e}_1^{\otimes j}\right)$ for all $j = 0, 1, 2, \cdots q$ is $O\left(\frac{q^2}{\epsilon^4}\log^4\frac{1}{\delta} + \frac{q^{3/2}}{\epsilon}\log\frac{1}{\delta}\cdot nnz(x)\right)$, where $\mathbf{e}_1 \in \mathbb{R}^d$ is the standard basis vector along the first coordinate.*

We prove this lemma in Appendix D. Now we are ready to design the procedure RowSampler to perform row norm sampling on matrices of the form $X^{\otimes q}(B^\top B+\lambda I)^{-1/2}$.

Overview of Algorithm 2: The goal is to generate a sample $(i_1, i_2, \cdots i_q) \in [d]^q$ with probability proportional to the squared norm of the row $(i_1, \cdots i_q)$ of the matrix $X^{\otimes q}(B^\top B+\lambda I)^{-1/2}$. Because the matrix $(B^\top B+\lambda I)^{-1/2}$ is of a large $n\times n$ size, we seek to compress it without perturbing the row norm distribution of $X^{\otimes q}(B^\top B+\lambda I)^{-1/2}$. This can be done by applying a JL-transformation to the rows of this matrix (see, e.g., (Dasgupta & Gupta, 2003; Kane & Nelson, 2014)). Let $H \in \mathbb{R}^{n\times d'}$ be a random matrix with i.i.d. normal entries with $d' = C_1 q\log_2 n$. Then with probability $1-\frac{1}{\text{poly}(n^q)}$ the norm of each row of the matrix $X^{\otimes q}(B^\top B+\lambda I)^{-1/2}\cdot H$ will be preserved up to a (1 ± 0.1) factor and hence by a union bound, with high probability all row norms of $X^{\otimes q}(B^\top B+\lambda I)^{-1/2}\cdot H$ are within a (1 ± 0.1) factor of the row norms of the original matrix. This is done in line 2 of the algorithm by computing the matrix $M = (B^\top B+\lambda I)^{-1/2}\cdot H$, which can be done quickly since B is a low rank matrix and H has few columns.

Now the problem is reduced to performing row norm sampling on $X^{\otimes q}M$. In order to generate a sample with distribution proportional to the squares of the row norms of $X^{\otimes q}M$ we can first sample a column of this matrix with probability proportional to the squared column norms and then generate a row index with probability proportional to the squared value of the entries of the selected column. This process generates a random index with our desired distribution. Computing the exact column norms of $X^{\otimes q}M$ is too expensive as this matrix has d^q rows, but if we apply the sketch Q^q from Lemma 4, we can compress the rows while preserving the column norms, in near input sparsity time, up to small error. So, it is enough to sample a column j with probability proportional to the squared column norms of $Q^q X^{\otimes q}M$, which is done in lines 3-7 of the algorithm.

Given that the j^{th} column of $X^{\otimes q}M$ was sampled, all we

Algorithm 2 RowSampler for Polynomial Kernel

input: $X \in \mathbb{R}^{d\times n}$, $q \in \mathbb{Z}_+$, $B \in \mathbb{R}^{m\times n}$, $\lambda \in \mathbb{R}_+$, $s \in \mathbb{Z}_+$
output: Sampling matrix $S \in \mathbb{R}^{s\times d^q}$

1: Generate $H \in \mathbb{R}^{n\times d'}$ with i.i.d. normal entries with $d' = C_1 q\log_2 n$
2: $M \leftarrow (B^\top B+\lambda I)^{-1/2}\cdot H$
3: Let $Q^q \in \mathbb{R}^{m'\times d^q}$ be an instance of the sketch from Lemma 4 with $\epsilon = \frac{1}{10q}$, $\delta = \frac{1}{\text{poly}(n)}$, $m' = C_2 q^2\log_2 n$
4: Compute $P_j = Q^q\left(X^{\otimes(q-j)}\otimes E_1^{\otimes j}\right)$ for all $j = 0, 1, \cdots q-1$, where $E_1 \in \mathbb{R}^{d\times n}$ is a matrix whose columns are copies of $\mathbf{e}_1$, i.e., $E_1 = \Big[\underbrace{\mathbf{e}_1, \mathbf{e}_1, \cdots \mathbf{e}_1}_{n \text{ copies}}\Big]$
5: $Z \leftarrow P_0 M$
6: $p_i \leftarrow \frac{\|Z_{\star,i}\|_2^2}{\|Z\|_F^2}$ for every $i \in [d']$
7: Generate i.i.d. samples $j_1, j_2, \cdots j_s \in [d']$ with distribution $\{p_i\}_{i=1}^{d'}$
8: Let $h : [d] \to [s']$ be a fully independent and uniform hash function with $s' = \lceil q^{3/2}s\rceil$
9: Let $h^{-1}(r) = \{j \in [d] : h(j) = r\}$ for every $r \in [s']$
10: For every $r \in [s']$, generate $G_r \in \mathbb{R}^{n'\times d_r}$ with i.i.d. normal entries where $d_r = |h^{-1}(r)|$ and $n' = C_3 q^2\log_2 n$
11: $W_r \leftarrow G_r \cdot X_{h^{-1}(r),\star}$ for every $r \in [s']$
12: **for** $l = 1$ to s **do**
13: $\quad D^{(0)} \leftarrow \text{diag}(M_{\star,j_l})$
14: $\quad$ **for** $a = 1$ to q **do**
15: $\quad\quad p_r^a \leftarrow \frac{\left\|W_r\cdot D^{(a-1)}\cdot P_a^\top\right\|_F^2}{\sum_{t=1}^{s'}\left\|W_t\cdot D^{(a-1)}\cdot P_a^\top\right\|_F^2}$ for every $r \in [s']$
16: $\quad\quad$ Generate a sample t with distribution $\{p_r^a\}_{r=1}^{s'}$
17: $\quad\quad q_i^a \leftarrow \frac{\left\|X_{i,\star}D^{(a-1)}P_a^\top\right\|_2^2}{\left\|X_{h^{-1}(t),\star}D^{(a-1)}P_a^\top\right\|_F^2}$ for every $i \in h^{-1}(t)$
18: $\quad\quad$ Sample an i_a with distribution $\{q_i^a\}_{i\in h^{-1}(t)}$
19: $\quad\quad D^{(a)} \leftarrow D^{(a-1)}\cdot\text{diag}(X_{i_a,\star})$
20: $\quad$ **end for**
21: $\quad \beta \leftarrow 0$
22: $\quad$ **for** $j = 1$ to d' **do**
23: $\quad\quad L^{(0)} \leftarrow \text{diag}(M_{\star,j})$
24: $\quad\quad$ **for** $b = 1$ to q **do**
25: $\quad\quad\quad p_b^* \leftarrow \frac{\left\|W_{h(i_b)}\cdot L^{(b-1)}\cdot P_b^\top\right\|_F^2}{\sum_{t=1}^{s'}\left\|W_t\cdot L^{(b-1)}\cdot P_b^\top\right\|_F^2}$
26: $\quad\quad\quad q_b^* \leftarrow \frac{\left\|X_{i_b,\star}L^{(b-1)}P_b^\top\right\|_2^2}{\left\|X_{h^{-1}(h(i_b)),\star}L^{(b-1)}P_b^\top\right\|_F^2}$
27: $\quad\quad\quad L^{(b)} \leftarrow L^{(b-1)}\cdot\text{diag}(X_{i_b,\star})$
28: $\quad\quad$ **end for**
29: $\quad\quad \beta \leftarrow \beta + sp_j\cdot\prod_{b=1}^q(p_b^* q_b^*)$
30: $\quad$ **end for**
31: $\quad$ Let l^{th} row of S be $\beta^{-1/2}\left(\mathbf{e}_{i_1}\otimes\mathbf{e}_{i_2}\otimes\cdots\mathbf{e}_{i_q}\right)^\top$
32: **end for**
33: **return** S

need to do is sample an entry of $X^{\otimes q}M_{\star,j}$ with probability proportional to the squared values of its entries. Note that forming this vector is out of the question since it has d^q coordinates. By basic properties of tensor products, the entries of $X^{\otimes q}M_{\star,j}$ are in bijective correspondence with the entries of the matrix $X \cdot \mathrm{diag}(M_{\star,j}) \cdot \left(X^{\otimes q-1}\right)^\top$, where entry $(i_1, i_2, \cdots i_q)$ of $X^{\otimes q}M_{\star,j}$ is equal to the entry at row i_1 and column $(i_2, \cdots i_q)$ of $X \cdot \mathrm{diag}(M_{\star,j}) \cdot \left(X^{\otimes q-1}\right)^\top$. Therefore, it is enough to sample an entry of the matrix $X \cdot \mathrm{diag}(M_{\star,j}) \cdot \left(X^{\otimes q-1}\right)^\top$ with probability proportional to its squared value. To this end, we first sample a row of this matrix with probability proportional to the squared row norms, and then sample a column by performing ℓ_2-sampling on the sampled row. Since $X \cdot \mathrm{diag}(M_{\star,j}) \cdot \left(X^{\otimes q-1}\right)^\top$ has a large number d^{q-1} of columns, we first sketch the rows of this matrix, incurring only a factor $\left(1 \pm \frac{1}{10q}\right)$ perturbation to the row norms, and then perform row norm sampling on the sketched matrix. Now we have an index $i_1 \in [d]$ sampled from the right distribution and all that is left to do is to carry out ℓ_2-sampling on the vector $X_{i_1,\star} \cdot \mathrm{diag}(M_{\star,j}) \cdot \left(X^{\otimes q-1}\right)^\top$. Note that we have made progress because this vector has size d^{q-1} and we have reduced the size by a factor of d. We recursively repeat this process of reshaping the tensor product to a matrix and sampling a row of the matrix q times until having all q indices $i_1, i_2, \cdots i_q$. Algorithm 2 does this. Note that the actual procedure requires more work because we need to generate s i.i.d. samples with the row norm distribution and to ensure that our runtime does not lose a multiplicative factor of s, resulting in $s \cdot \mathrm{nnz}(X)$ total time, we need to do extra sketching and a random partitioning of the rows of the matrix X to $\Theta(q^{3/2}s)$ buckets. The formal guarantee on Algorithm 2 is given in the following lemma.

Lemma 5. *For any matrices $X \in \mathbb{R}^{d\times n}$ and $B \in \mathbb{R}^{m\times n}$, any $\lambda > 0$ and any positive integers q, s, with high probability, Algorithm 2 outputs a ranks-s row norm sampler for $X^{\otimes q}(B^\top B + \lambda I)^{-1/2}$ (Definition 3.1) in time $O\left(m^2 n + q^{15/2}s^2 n \log^3 n + q^{5/2}\log^3 n \cdot nnz(X)\right)$.*

Proof. All rows of the sampling matrix $S \in \mathbb{R}^{s\times d^q}$ (output of Algorithm 2) have independent and identical distributions because the algorithm generates i.i.d. samples $j_1, j_2, \cdots j_s$ in line 7 and then for each $l \in [s]$, the l^{th} row of the matrix S is constructed by sampling $i_1, i_2, \cdots i_q$ in line 18 from a distribution that is solely determined by j_l and is independent of the values of $j_{l'}$ for $l' \neq l$.

Since every row of S is identically distributed, let us consider the distribution of the l^{th} row of S for some arbitrary $l \in [s]$. Let J be a random variable that takes values in $\{1, 2, \cdots d'\}$ with probability distribution $\{p_i\}_{i=1}^{d'}$ defined in line 6 of Algorithm 2. The random index j_l generated in line 7 of the algorithm is a copy of the random variable J.

For any $j \in [d']$, let $I^j = (I_1^j, I_2^j, \cdots I_q^j)$ be a vector-valued random variable that takes values in $[d]^q$ with the following conditional probability distribution for every $a = 1, 2, \cdots q$,

$$\Pr[I_a^j = i | I_1^j = i_1, I_2^j = i_2, \cdots I_{a-1}^j = i_{a-1}]$$
$$= \frac{\|W_{h(i)} \cdot D^{j,a-1}P_a^\top\|_F^2}{\sum_{t=1}^{s'}\|W_t D^{j,a-1}P_a^\top\|_F^2} \frac{\|X_{i,\star}D^{j,a-1}P_a^\top\|_2^2}{\|X_{h^{-1}(h(i)),\star}D^{j,a-1}P_a^\top\|_F^2},$$

where W_r for every $r \in [s']$ are the matrices defined in line 11 of the algorithm and $D^{j,a-1}$ is a diagonal matrix of size $n \times n$ whose diagonal entries are $D_{rr}^{j,a-1} = M_{r,j} \cdot \prod_{b=1}^{a-1} X_{i_b,r}$, for every $r \in [n]$ and $a \in [q]$. For ease of notation we drop the superscript j and just write D^{a-1}. One can verify that the vector random variable $(i_1, i_2, \cdots i_q)$ obtained by stitching together the random indices generated in line 18 of the algorithm, is a copy of the random variable I^{j_l}.

Let β be the quantity that the for loop in lines 21-30 of the algorithm computes. If $i_1, i_2, \cdots i_q \in [d]$ are the indices sampled in line 18 of the algorithm, then the value of β can be computed as, $\beta = s\sum_{j=1}^{d'} p_j \prod_{b=1}^{q} p_b^* q_b^*$, where $p_b^* = \frac{\|W_{h(i_b)} \cdot D^{b-1}P_b^\top\|_F^2}{\sum_{t=1}^{s'}\|W_t D^{b-1}P_b^\top\|_F^2}$ and $q_b^* = \frac{\|X_{i_b,\star}D^{b-1}P_b^\top\|_2^2}{\|X_{h^{-1}(h(i_b)),\star}D^{b-1}P_b^\top\|_F^2}$ are the quantities computed in lines 25 and 26 of the algorithm. Hence, for any $i_1, i_2, \cdots i_q \in [d]$, the distribution of $S_{l,\star}$ is,

$$\Pr\left[S_{l,\star} = \beta^{-1/2}(\mathbf{e}_{i_1} \otimes \mathbf{e}_{i_2} \otimes \cdots \mathbf{e}_{i_q})^\top\right]$$
$$= \sum_{j=1}^{d'} \Pr\left[S_{l,\star} = \beta^{-1/2}(\mathbf{e}_{i_1} \otimes \cdots \mathbf{e}_{i_q})^\top \middle| J = j\right] \cdot p_j$$
$$= \sum_{j=1}^{d'} \Pr\left[I^j = (i_1, i_2, \cdots i_q)\right] \cdot p_j. \tag{2}$$

By the law of total probability, we have $\Pr\left[I^j = (i_1, i_2, \cdots i_q)\right] = \prod_{a=1}^{q} \Pr[I_a^j = i_a | I_1^j = i_1, \cdots I_{a-1}^j = i_{a-1}]$, and therefore, because $\Pr\left[I_a^j = i_a | I_1^j = i_1, \cdots I_{a-1}^j = i_{a-1}\right] = p_a^* q_a^*$, we find that

$$\Pr\left[S_{l,\star} = \beta^{-1/2}(\mathbf{e}_{i_1} \otimes \mathbf{e}_{i_2} \otimes \cdots \mathbf{e}_{i_q})^\top\right] = \frac{\beta}{s}.$$

Now note that for any $r \in [s']$, W_r is defined as $W_r = G_r \cdot X_{h^{-1}(r),\star}$ where G_r is a matrix with i.i.d. normal entries with $n' = C_3 q^2 \log_2 n$ rows. Therefore, G_r is a JL-transform and for every $a \in [q], r \in [s']$, with high probability, i.e.,

$$\frac{\|W_r D^{a-1}P_a^\top\|_F^2}{n'} \in \frac{\|X_{h^{-1}(r),\star}D^{a-1}P_a^\top\|_F^2}{1 \pm 1/10q}. \tag{3}$$

For a simple proof of (3), see (Dasgupta & Gupta, 2003) (see also (Kane & Nelson, 2014) for a more efficient version). By

union bounding over $qs'd'$ events, (3) holds simultaneously for all $a \in [q]$, $j \in [d']$, $r \in [s']$ with high probability. We condition on (3) holding in what follows. We can bound the conditional probability of I_a^j as follows,

$$\Pr[I_a^j = i | I_1^j = i_1, I_2^j = i_2, \cdots I_{a-1}^j = i_{a-1}] \geq (1 - 1/5q) \cdot \frac{\|X_{i,\star} D^{a-1} P_a^\top\|_2^2}{\|X D^{a-1} P_a^\top\|_F^2}. \quad (4)$$

For every $a \in [q]$, line 4 of the algorithm computes $P_a = Q^q \left(X^{\otimes(q-a)} \otimes E_1^{\otimes a}\right)$, where Q^q is the sketch from Lemma 4 with $\epsilon = \frac{1}{10q}$. By basic properties of tensor products, for every $i \in [d]$,

$$P_a D^{a-1} X_{i,\star}^\top = Q^q \left(X^{\otimes(q-a)} \otimes E_1^{\otimes a}\right) D^{a-1} X_{i,\star}^\top = Q^q \left(\left(X^{\otimes(q-a)} D^{a-1} X_{i,\star}^\top\right) \otimes \mathbf{e}_1^{\otimes a}\right).$$

Hence, by Lemma 4, for every $a \in [q]$ and every $i \in [d]$, with high probability,

$$\left\|X_{i,\star} D^{a-1} P_a^\top\right\|_2^2 \in \frac{\left\|X^{\otimes(q-a)} D^{a-1} X_{i,\star}^\top\right\|_2^2}{1 \pm 0.1/q}. \quad (5)$$

By union bounding over $qd'd$ events, with high probability, (5) holds simultaneously for all $a \in [q]$, all $j \in [d']$, and all $i \in [d]$. Therefore, conditioning on (5) holding and using (4), the conditional probability of I_a^j satisfies

$$\Pr[I_a^j = i | I_1^j = i_1, I_2^j = i_2, \cdots I_{a-1}^j = i_{a-1}] \geq (1 - 2/5q) \frac{\|X^{\otimes(q-a)} D^{a-1} X_{i,\star}^\top\|_2^2}{\|X^{\otimes(q-a)} D^{a-1} X^\top\|_F^2}. \quad (6)$$

It follows from the definition of tensor products and definition of D^a, that

$$\left\|X^{\otimes(q-a-1)} D^a X^\top\right\|_F^2 = \left\|X^{\otimes(q-a-1)} D^{a-1} \cdot \mathrm{diag}(X_{i_a,\star}) X^\top\right\|_F^2 = \left\|X^{\otimes(q-a)} D^{a-1} X_{i_a,\star}^\top\right\|_2^2$$

Using this equality and inequality (6),

$$\begin{aligned}
&\Pr\left[I^j = (i_1, i_2, \cdots i_q)\right] \\
&= \prod_{a=1}^{q} \Pr\left[I_a^j = i_a | I_1^j = i_1, \cdots I_{a-1}^j = i_{a-1}\right] \\
&\geq \prod_{a=1}^{q} \left(1 - \frac{2}{5q}\right) \frac{\|X^{\otimes(q-a)} D^{a-1} X_{i_a,\star}^\top\|_2^2}{\|X^{\otimes(q-a)} D^{a-1} X^\top\|_F^2} \\
&\geq \frac{1}{2} \frac{\|X^{\otimes(0)} D^{q-1} X_{i_q,\star}^\top\|_2^2}{\|X^{\otimes(q-1)} D^0 X^\top\|_F^2} \\
&= \frac{1}{2} \frac{\left|[X^{\otimes q} \cdot M]_{(i_1,i_2,\cdots i_q),j}\right|^2}{\|[X^{\otimes q} \cdot M]_{\star,j}\|_2^2} \quad (7)
\end{aligned}$$

By plugging (7) back in (2) we find that,

$$\begin{aligned}
&\Pr\left[S_{l,\star} = \beta^{-1/2} (\mathbf{e}_{i_1} \otimes \mathbf{e}_{i_2} \otimes \cdots \mathbf{e}_{i_q})^\top\right] \\
&\geq \sum_{j=1}^{d'} \frac{1}{2} \cdot \frac{\left|[X^{\otimes q} \cdot M]_{(i_1,i_2,\cdots i_q),j}\right|^2}{\|[X^{\otimes q} \cdot M]_{\star,j}\|_2^2} \cdot p_j \\
&= \frac{1}{2} \sum_{j=1}^{d'} \frac{\left|[X^{\otimes q} \cdot M]_{(i_1,i_2,\cdots i_q),j}\right|^2}{\|X^{\otimes q} M_{\star,j}\|_2^2} \frac{\|Q^q X^{\otimes q} M_{\star,j}\|_2^2}{\|Q^q X^{\otimes q} M\|_F^2} \\
&\geq \frac{1}{3} \sum_{j=1}^{d'} \frac{\left|[X^{\otimes q} \cdot M]_{(i_1,i_2,\cdots i_q),j}\right|^2}{\|X^{\otimes q} M_{\star,j}\|_2^2} \frac{\|X^{\otimes q} M_{\star,j}\|_2^2}{\|X^{\otimes q} M\|_F^2} \\
&= \frac{1}{3} \frac{\left\|[X^{\otimes q} \cdot M]_{(i_1,i_2,\cdots i_q),\star}\right\|^2}{\|X^{\otimes q} M\|_F^2}
\end{aligned}$$

Matrix M is defined as $M = (B^\top B + \lambda I)^{-1/2} \cdot H$ where H is a random matrix with i.i.d. Gaussian entries with $d' = C_1 q \log_2 n$ columns. Therefore, H is a JL-transform, so for every $(i_1, i_2, \cdots i_q) \in [d]^q$, with probability $1 - \frac{1}{\mathrm{poly}(n^q)}$,

$$\frac{\left\|[X^{\otimes q} \cdot M]_{(i_1,i_2,\cdots i_q),\star}\right\|_2^2}{d'} \in (1 \pm 0.1) \left\|[X^{\otimes q}]_{(i_1,i_2,\cdots i_q),\star} (B^\top B + \lambda I)^{-1/2}\right\|_2^2.$$

Therefore, by union bounding over d^q rows of $X^{\otimes q} M$, the above holds simultaneously for all $(i_1, i_2, \cdots i_q) \in [d]^q$ with high probability. Therefore,

$$\Pr\left[S_{l,\star} = \beta^{-1/2} (\mathbf{e}_{i_1} \otimes \mathbf{e}_{i_2} \otimes \cdots \mathbf{e}_{i_q})^\top\right] \geq \frac{1}{4} \cdot \frac{\left\|\left[X^{\otimes q} \cdot (B^\top B + \lambda I)^{-1/2}\right]_{(i_1,i_2,\cdots i_q),\star}\right\|_2^2}{\|X^{\otimes q} (B^\top B + \lambda I)^{-1/2}\|_F^2}$$

Because $\frac{\beta}{s}$ is the probability of sampling row $(i_1, i_2, \cdots i_q)$ of the matrix $X^{\otimes q}(B^\top B + \lambda I)^{-1/2}$, the above inequality proves that with high probability, S is a rank-s row norm sampler for $X^{\otimes q}(B^\top B + \lambda I)^{-1/2}$ as in Definition 3.1.

Runtime: One of the expensive steps of this algorithm is the computation of M in line 2 which takes $O(m^2 n + qmn \log n)$ operations since B is rank m. Another expensive step is the computation of P_j for $j = 0, 1, \cdots q-1$ in line 4. By Lemma 4, this can be computed in time $O\left(q^6 n \log^4 n + q^{5/2} \log n \cdot \mathrm{nnz}(X)\right)$. Matrices W_r for all $r \in [s']$ in line 11 of the algorithm can be computed in time $O(q^2 \log n \cdot \mathrm{nnz}(X))$. Computing the distribution $\{p_r^a\}_{r=1}^{s'}$ in line 15 takes time $O\left(q^{11/2} sn \log^2 n\right)$ for a fixed $a \in [q]$ and $l \in [s]$. Therefore, the total time to compute this distribution for all a and l is $O\left(q^{13/2} s^2 n \log^2 n\right)$.

The runtime to compute the distribution $\{q_i^q\}_{i \in h^{-1}(t)}$ in line 17 depends on the sparsity of $X_{h^{-1}(t),\star}$, i.e.,

$\text{nnz}(X_{h^{-1}(t),\star})$. To bound the sparsity of $X_{h^{-1}(t),\star}$, note that, $\text{nnz}(X_{h^{-1}(t),\star}) = \sum_{i=1}^{d} \mathbf{1}_{\{i\in h^{-1}(t)\}} \cdot \text{nnz}(X_{i,\star})$. Let us introduce the random variables $S_1, S_2, \cdots S_d$ defined as $S_i = \left(\mathbf{1}_{\{i\in h^{-1}(t)\}} - \frac{1}{s'}\right) \cdot \text{nnz}(X_{i,\star})$ for $i \in [d]$. Since the hash function h is fully independent, the random variables $S_1, S_2, \cdots S_d$ are independent. Also, each of these random variables is zero mean and uniformly bounded, i.e., $\mathbb{E}[S_i] = 0$ and $|S_i| \leq n$ for each $i \in [d]$. Therefore we can invoke Bernstein's inequality (Appendix A). Let $Z = \sum_{i=1}^{d} S_i$. Then the variance of the sum is bounded as $\sum_{i=1}^{d} \mathbb{E}[S_i^2] \leq \sum_{i=1}^{d} \frac{1}{s'}\text{nnz}(X_{i,\star})^2 \leq \frac{n}{s'} \cdot \text{nnz}(X)$.

By invoking Bernstein's inequality, for some constant C, $\Pr\left[|Z| \geq C\log_2 n \cdot \left(\sqrt{\frac{n}{s'} \cdot \text{nnz}(X)} + n\right)\right] \leq \frac{1}{\text{poly}(n)}$. Hence, for every $t \in [s']$, with high probability $\text{nnz}(X_{h^{-1}(t),\star}) = O\left((\text{nnz}(X)/s' + n)\log n\right)$. By union bounding over s' events, with high probability, $\text{nnz}(X_{h^{-1}(t),\star}) = O\left((\text{nnz}(X)/s' + n)\log n\right)$, simultaneously for all $t \in [s']$ which implies that the distribution $\{q_i^q\}_{i\in h^{-1}(t)}$ in line 17 of the algorithm can be computed in time $O\left(q^2 n\log^2 n + q^2\log^2 n \cdot \text{nnz}(X)/s'\right)$ for a fixed $a \in [q]$ and a fixed $l \in [s]$. Therefore the total time to compute this distribution for all a and all l is $O\left(q^3 sn\log^2 n + q^{3/2}\log^2 n \cdot \text{nnz}(X)\right)$.

Finally the last expensive step is the computation of quantities p_b^* and q_b^* in lines 25 and 26 of the algorithm. Both of these quantities can be computed in time $O\left(q^{11/2} sn\log^2 n + q^2\log^2 n \cdot \text{nnz}(X)/s'\right)$ for a fixed $j \in [d']$ and a fixed $b \in [q]$. Therefore the total time to compute these quantities for all l, all j, and all b is $O\left(q^{15/2}s^2 n\log^3 n + q^{5/2}\log^3 n \cdot \text{nnz}(X)\right)$. Therefore the total runtime of Algorithm 2 is $O\left(m^2 n + q^{15/2}s^2 n\log^3 n + q^{5/2}\log^3 n \cdot \text{nnz}(X)\right)$. □

We prove Theorem 1 in Appendix E.

3.2. Adaptive Sampling for the Gaussian Kernel

Consider the lifting corresponding to the Gaussian kernel, $k(x,y) = e^{-\|x-y\|_2^2/2}$, that can be obtained through a Taylor expansion. This feature mapping was exploited in (Ahle et al., 2020) to obtain an efficient subspace embedding for the Gaussian kernel via sketching the polynomial terms in its Taylor expansion. For datasets with bounded radius, the Gaussian kernel can be well-approximated by a superposition of low-degree polynomial kernels. We formally define this approximate feature mapping (lifting) as follows.

Definition 3.2 (Polynomial Lifting for Gaussian Kernel)**.** For any integer q the *degree-q polynomial lifting for Gaussian kernel* is the mapping $\phi_q : \mathbb{R}^d \to \mathbb{R}^D$, defined as,

$$\phi_q(x) = e^{-\|x\|_2^2/2}\left(\frac{x^{\otimes 0}}{\sqrt{0!}} \oplus \frac{x^{\otimes 1}}{\sqrt{1!}} \oplus \frac{x^{\otimes 2}}{\sqrt{2!}} \oplus \cdots \frac{x^{\otimes q}}{\sqrt{q!}}\right),$$

for $x \in \mathbb{R}^d$, where $D = \sum_{j=0}^{q} d^j$.

Claim 6. Let $x_1, x_2, \cdots x_n \in \mathbb{R}^d$ be a dataset with bounded radius, i.e., $\|x_i\|_2^2 \leq r$ for all $i \in [n]$. Suppose $K \in \mathbb{R}^{n\times n}$ is the Gaussian kernel corresponding to this dataset ($K_{i,j} = e^{-\|x_i - x_j\|_2^2/2}$). Also suppose that ϕ_q is the degree-q polynomial lifting for the Gaussian kernel as in Definition 3.2. If A is a matrix with n columns whose columns are obtained by applying the map ϕ_q on the data points, i.e., $A_{\star,i} = \phi_q(x_i)$, then as long as $q = \Omega\left(r + \log n\right)$, we have $\|A^\top A - K\|_{op} \leq \frac{1}{\text{poly}(n)}$.

Therefore, to find a spectral approximation to the Gaussian kernel K for bounded datasets, it is enough to find a spectral approximation to $A^\top A$, where A is the matrix defined in the above claim. We have designed an efficient adaptive sampling method for tensor products of the form $X^{\otimes j}$ in the previous section. Since matrix A is a concatenation of tensor products $X^{\otimes j}$ for $j = 0, 1, \cdots q$, using our iterative leverage score sampling procedure for the polynomial kernel we can spectrally approximate $A^\top A$ in nearly $\text{nnz}(X)$ time. We present a full algorithm which can perform recursive leverage score sampling on matrix A and analyze it in Appendix F and prove Theorem 2 in Appendix G.

3.3. Generalization to dot-product Kernels

An important technical contribution of this paper is a sampling method that can embed the polynomial kernel using near-optimal runtime. Additionally, our method can be used for embedding a wide class of kernels that can be well-approximated by low-degree polynomials. In particular, our sampling method can be applied to any dot-product kernel with a rapidly convergent Taylor expansion. In this section, we argue how our method can be generalized to such kernels.

The underlying observation that enables us to extend our subspace embedding to the class of dot-product kernels is a classical result in harmonic analysis due to Schoenberg (1988), that characterizes positive definite functions in a Hilbert space. This observation is simply the fact that any dot-product kernel $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ defined as $k(x,y) = f(\langle x, y\rangle)$ must have a Taylor expansion with only non-negative coefficients, i.e., k is a kernel function if and only if $f(\alpha) = \sum_{j=0}^{\infty} a_j\alpha^j$, $a_j \geq 0$ for all $j \in \mathbb{Z}$. As a result, truncating this sum at any point results in a valid kernel, that is $k_q(x,y) := \sum_{j=0}^{q} a_j\langle x, y\rangle^j$ is a valid positive definite kernel.

For most dot-product kernels used in practice, the coefficients a_j decay at least exponentially. If this is the case, then $|k_q(x,y) - k(x,y)| \leq \frac{1}{\text{poly}(n)}$ for any $x, y \in \mathbb{R}^d$ with $\|x\|_2^2, \|y\|_2^2 \leq r$ and $q = \Omega(r\log n)$. Hence, in order to obtain a subspace embedding for kernel k on any dataset with bounded ℓ_2 radius, it is enough to find a subspace embed-

ding for the truncated kernel $k_q(x,y) = \sum_{j=0}^{q} a_j \langle x,y\rangle^j$. Since this kernel is a superposition of polynomial kernels, we can apply our subspace embedding for the polynomial kernel from Section 3.1 to each of the polynomial terms. This will result in a near input sparsity time subspace embedding for any dot-product kernel whose Taylor expansion decays at least exponentially.

An example of a well known dot product kernel is the inverse polynomial kernel defined as $k(x,y) = \frac{1}{2-\langle x,y\rangle}$. The Taylor expansion of this kernel is $k(x,y) = \sum_{j=0}^{\infty} 2^{-j-1}\langle x,y\rangle^j$. Therefore, if we let $q = \Theta(\log n)$ then for any $x, y \in \mathbb{R}^d$ with $\|x\|_2^2, \|y\|_2^2 \leq 1$, $|k_q(x,y) - k(x,y)| \leq \frac{1}{\text{poly}(n)}$, where $k_q(x,y) = \sum_{j=0}^{q} 2^{-j-1}\langle x,y\rangle^j$. Hence, we can obtain a subspace embedding for the inverse polynomial kernel in nearly $\text{nnz}(X)$ time by applying our sampling method from Section 3.1 to polynomials of degree $O(\log n)$ in this Taylor expansion.

4. Experiments

In this section we assess the performance of our result for embedding the Gaussian kernel (Theorem 2) against the Fourier features (**FF**) method (Rahimi & Recht, 2008), **Nystrom** method (Musco & Musco, 2017), as well as the **Oblivious** sketching method of (Ahle et al., 2020). The results are summarized in Table 1[2]. Our importance sampling algorithm is a recursive procedure given in Algorithm 1. In this set of experiments, we also consider a variant of our sampling algorithm that runs only a single round of the recursive sampling and hence is considerably faster. This variant is equivalent to sampling rows of the lifting matrix Φ with probabilities proportional to the squared row norms. We denote this variant of our method by **Row norm** and denote the full recursive importance sampling algorithm by **Adaptive**. The target dimension of all methods is denoted by s in Table 1.

We base our comparison on the four standard large-scale regression datasets evaluated in (Le et al., 2013). The size of the data points is denoted by n and the dimensionality is denoted by d in Table 1. In all experiments, we first find a low-rank approximation to the kernel matrix using various feature sampling/sketching techniques. Then, using the kernel's proxy, we find an approximate regressor by solving an ℓ_2 regularized least-squares problem. For all methods, Table 1 reports the total time to train the regressors, including the runtime of feature sampling and the runtime of linear regression. We use the same hyperparameters (kernel bandwidth and regularization parameter) across all kernel approximation methods which were selected via cross-validation on the Fourier features method, as our baseline method. For every method, we set the number s of features to the smallest value such that increasing the number of features does not improve the error non-negligibly.

[2]We repeated the experiments with 5 different random seeds and reported the average RMSE and runtime in Table 1.

Table 1. The RMSE on the test set along with the total training time of approximate KRR via various approximation methods.

DATASET:	WINE $n = 6,497$ $d = 11$	INSURANCE $n = 9,822$ $d = 85$	CT LOCATION $n = 53,500$ $d = 384$	FOREST $n = 581,012$ $d = 54$
FF	0.736, 2 SEC $s = 5000$	0.231, 1 SEC $s = 2000$	3.89, 1 MIN $s = 4000$	1.00, 3 MIN $s = 1000$
Nystrom	0.730, 1.5 MIN $s = 2000$	0.231, 1.5 MIN $s = 2000$	3.86, 8.5 MIN $s = 1500$	1.03, 8 MIN $s = 500$
Oblivious	0.732, 13 SEC $s = 1024$	0.231, 20 SEC $s = 1024$	3.70, 3.5 MIN $s = 5120$	1.05, 2.5 MIN $s = 320$
Row norm	0.727, 3 SEC $s = 5000$	0.231, 2 SEC $s = 1500$	3.68, 1 MIN $s = 6000$	1.08, 2.5 MIN $s = 1000$
Adaptive	0.723, 15 SEC $s = 400$	0.232, 6 SEC $s = 400$	3.72, 8.5 MIN $s = 2800$	1.05, 7 MIN $s = 500$

The *Row norm* variant of our method is as fast as the *FF* method and runs significantly faster than the *Nystrom* and *Oblivious* methods while having superior testing RMSE. Our full algorithm, *Adaptive*, has even better performance than our single round variant *Row norm* in terms of RMSE on the test set and achieves a better RMSE while having a significantly smaller target dimension s than all other methods. In terms of runtime, our full *Adaptive* method is no worse than *Nystrom* but is slower than our single round *Row norm* method. Our *Adaptive* method has a slightly better RMSE than the *Oblivious* method and runs slower, but it achieves a significantly smaller target dimension s. However, our single round *Row norm* variant is significantly faster than *Oblivious*.

Acknowledgements

D. P. Woodruff was supported in part by Office of Naval Research (ONR) grant N00014-18-1-2562.

References

Ahle, T. D., Kapralov, M., Knudsen, J. B., Pagh, R., Velingker, A., Woodruff, D. P., and Zandieh, A. Oblivious sketching of high-degree polynomial kernels. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 141–160. SIAM, 2020.

Alaoui, A. and Mahoney, M. W. Fast randomized kernel ridge regression with statistical guarantees. In *Advances in Neural Information Processing Systems*, pp. 775–783, 2015.

Avron, H., Nguyen, H., and Woodruff, D. Subspace embeddings for the polynomial kernel. In *Advances in neural information processing systems*, pp. 2258–2266, 2014.

Avron, H., Clarkson, K. L., and Woodruff, D. P. Faster kernel ridge regression using sketching and preconditioning. *SIAM Journal on Matrix Analysis and Applications*, 38 (4):1116–1138, 2017a.

Avron, H., Kapralov, M., Musco, C., Musco, C., Velingker, A., and Zandieh, A. Random fourier features for kernel ridge regression: Approximation bounds and statistical guarantees. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 253–262. JMLR. org, 2017b.

Cohen, M. B., Musco, C., and Pachocki, J. Online row sampling. *arXiv preprint arXiv:1604.05448*, 2016.

Cohen, M. B., Musco, C., and Musco, C. Input sparsity time low-rank approximation via ridge leverage score sampling. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 1758–1777. SIAM, 2017.

Dasgupta, S. and Gupta, A. An elementary proof of a theorem of johnson and lindenstrauss. *Random Structures & Algorithms*, 22(1):60–65, 2003.

Kane, D. M. and Nelson, J. Sparser johnson-lindenstrauss transforms. *Journal of the ACM (JACM)*, 61(1):4, 2014.

Kapralov, M., Lee, Y. T., Musco, C., Musco, C., and Sidford, A. Single pass spectral sparsification in dynamic streams. In *2014 IEEE 55th Annual Symposium on Foundations of Computer Science*, pp. 561–570. IEEE, 2014.

Le, Q., Sarlós, T., and Smola, A. Fastfood-approximating kernel expansions in loglinear time. In *Proceedings of the international conference on machine learning*, volume 85, 2013.

Musco, C. and Musco, C. Recursive sampling for the nystrom method. In *Advances in Neural Information Processing Systems*, pp. 3833–3845, 2017.

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

Rahimi, A. and Recht, B. Random features for large-scale kernel machines. In *Advances in neural information processing systems*, pp. 1177–1184, 2008.

Schoenberg, I. Positive definite functions on spheres. *Duke Math. J*, 1:172, 1988.

Zandieh, A., Nouri, N., Velingker, A., Kapralov, M., and Razenshteyn, I. Scaling up kernel ridge regression via locality sensitive hashing. In *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, volume 108 of *Proceedings of Machine Learning Research*, pp. 4088–4097, Online, 26–28 Aug 2020. PMLR.