Improved Generalization Bound and Learning of Sparsity Patterns for Data-Driven Low-Rank Approximation

Shinsaku Sakaue
The University of Tokyo

Taihei Oki
The University of Tokyo

Abstract

Learning sketching matrices for fast and accurate low-rank approximation (LRA) has gained increasing attention. Recently, Bartlett, Indyk, and Wagner (COLT 2022) presented a generalization bound for the learning-based LRA. Specifically, for rank-$k$ approximation using an $m \times n$ learned sketching matrix with $s$ non-zeros in each column, they proved an $\tilde{O}(nsm)$ bound on the fat shattering dimension ($\tilde{O}$ hides logarithmic factors). We build on their work and make two contributions.

(1) We present a better $\tilde{O}(nsk)$ bound ($k \leq m$). En route to obtaining this result, we give a low-complexity Goldberg–Jerrum algorithm for computing pseudo-inverse matrices, which would be of independent interest.

(2) We alleviate an assumption of the previous study that sketching matrices have a fixed sparsity pattern. We prove that learning positions of non-zeros increases the fat shattering dimension only by $O(ns \log n)$. In addition, experiments confirm the practical benefit of learning sparsity patterns.

1 INTRODUCTION

Low-rank approximation (LRA) has played a crucial role in analyzing matrix data. Although the singular value decomposition (SVD) provides an optimal LRA, it is too costly when the data size is huge. To overcome this limitation, researchers have developed fast LRA methods with sketching, whose basic form is as follows: given an input matrix $A \in \mathbb{R}^{n \times d}$ and a target low rank $k$, choose a sketching matrix $S \in \mathbb{R}^{m \times n}$ with $k \leq m \leq \min\{n, d\}$ and compute an LRA matrix for $SA \in \mathbb{R}^{m \times d}$. If $S$ is drawn from an appropriate distribution, the resulting matrix is a good LRA of $A$ with high probability (Sarlos, 2006; Clarkson and Woodruff, 2009, 2017). This randomized sketching paradigm has led to various time- and space-efficient algorithms in numerical linear algebra. We refer the reader to (Woodruff, 2014; Martinsson and Tropp, 2020) for more details on this area.

While such LRA methods with randomized sketching enjoy rigorous guarantees even for worst-case input matrices, a recent line of work (Indyk et al., 2019; Liu et al., 2020; Indyk et al., 2021) suggests that learning-based LRA methods can attain significantly smaller approximation errors when we can use past data to better handle future data. They have achieved fast and more accurate LRA by learning sketching matrices $S$ to minimize approximation errors over past data.

As for the theoretical side of learning-based LRA, Bartlett et al. (2022) recently presented generalization bounds for learning sketching matrices. Specifically, they proved an $O(nsm)^3$ upper bound on the fat shattering dimension for learning an $m \times n$ sketching matrix with $s$ non-zeros at fixed positions in each column. They also showed an $\Omega(ns)$ lower bound. We give an overview of their work in Section 2.3.

Their study has raised some natural questions. For example, can we narrow the $O(m)$ gap between the upper and lower bounds? Moreover, generalization bounds for learning-based LRA with changeable sparsity patterns are awaited since learning positions of non-zeros is considered to be a promising direction (Indyk et al., 2021) and its effectiveness has been partly confirmed (Liu et al., 2020).

1.1 Our Contribution

Building on (Bartlett et al., 2022), we address the aforementioned questions and make two contributions.

First, we improve the previous $\tilde{O}(nsm)$ upper bound by replacing the $O(m)$ factor with $O(\log m)$, which leads to a better $\tilde{O}(nsk)$ bound ($k \leq m$). Although the sketching dimension, $m$, is often set to, for example, $4k$ in practice, there is no such theoretical relation as $m = O(k)$. Thus, our bound indeed improves the previous one. We take the same proof strategy as (Bartlett et al., 2022) and represent computational procedures of a loss function by a Goldberg–Jerrum

---

1 We use $\tilde{O}$ and $\tilde{\Omega}$ to hide logarithmic factors.
(GJ) algorithm. Our technical contribution is to develop a new GJ algorithm for computing pseudo-inverse matrices with a smaller predicate complexity than the previous one, which would be of independent interest. To demonstrate its usefulness, we also give a generalization bound for a learning-based Nyström method by using our GJ algorithm.

Second, we give a generalization bound for learning-based LRA with changeable sparsity patterns. Supposing we can learn both positions and values of \( ns \) non-zeros in a sketching matrix \( S \), we prove that our \( O(nsk) \) upper bound on the fat shattering dimension increases only by \( O(ns \log n) \), despite the presence of exponentially many possible sparsity patterns in \( ns \). Hence, the bound remains \( O(nsk) \) (ignoring \( O(\log n) \)) even when the sparsity pattern can change. Also, experiments show that a recent efficient learning-based LRA method (Indyk et al., 2021), which used fixed sparsity patterns, can achieve higher accuracy with changeable sparsity patterns, suggesting the practical benefit of our result.

1.2 Related Work

The most relevant study to ours is (Bartlett et al., 2022). They proved generalization bounds for learning-based LRA and other methods in numerical linear algebra. Other theoretical results related to learning-based LRA include safeguard guarantees (Indyk et al., 2019) and consistency (Indyk et al., 2021), which are different from generalization guarantees, as mentioned in (Bartlett et al., 2022, Section 2.5).

Gupta and Roughgarden (2017) initiated the study of a PAC-learning approach to algorithm configuration, which is also called data-driven algorithm design (Balcan, 2021). Recent studies have presented generalization bounds for various learning-based algorithms, e.g., integer programming methods (Balcan et al., 2018, 2021b, 2022), clustering (Balcan et al., 2020a), and heuristic search (Sakaue and Oki, 2022). Balcan et al. (2021a) presented a general theory for deriving generalization bounds based on piecewise structures of dual function classes. Their idea, however, does not lead to strong guarantees in learning-based LRA, as discussed in (Bartlett et al., 2022, Appendix E). As with (Bartlett et al., 2022), we consider a class of proxy loss functions to obtain a generalization bound. This idea has a slight connection to (Balcan et al., 2020b), which approximates dual functions with simpler ones, while the technical details are different.

2 BACKGROUND

For any positive integer \( n \), let \([n] = \{1, \ldots, n\}\). Let \( \text{sign}(\cdot) \) be the sign function that takes \( x \in \mathbb{R} \) as input and returns +1 if \( x > 0 \), −1 if \( x < 0 \), or 0 if \( x = 0 \). We define the degree of a polynomial by its total degree. The degree of a rational function refers to the maximum of its numerator’s and denominator’s degrees, where the fraction is reduced to the lowest terms.

Let \( \operatorname{rank}, \operatorname{tr}, \) and \( \operatorname{det} \) denote the rank, trace, and determinant. Let \( \|A\|_F^2 = \sqrt{\operatorname{tr}(A^\top A)} \) denote the Frobenius norm of a matrix \( A \). The Moore–Penrose pseudo-inverse of a matrix \( A \) is denoted by \( A^+ \). SVD refers to the compact singular value decomposition, i.e., for \( A \in \mathbb{R}^{n \times d} \) with \( \operatorname{rank}(A) = r \), SVD computes \( U \in \mathbb{R}^{n \times r}, \Sigma \in \mathbb{R}^{r \times r}, \) and \( V \in \mathbb{R}^{d \times r} \) with \( A = U \Sigma V^\top \). For any vector \( x \in \mathbb{R}^n \), let \( \supp(x) \subseteq [n] \) denote the set of indices of non-zeros. A sparsity pattern, \( J \subseteq [n] \), of \( x \) indicates that \( x_i \) is allowed to be non-zero if and only if \( i \in J \), hence \( \supp(x) \subseteq J \).

2.1 Learning Theory

Let \( X \) be a domain of inputs, \( \mathcal{D} \) a distribution over \( X \), and \( \mathcal{L} \subseteq [0,1]^X \) a class of loss functions. In our case, \( X \) is a class of input matrices, and each \( L \in \mathcal{L} \) measures the approximation error of LRA and is parametrized by a sketching matrix (see Section 2.3). For \( \delta \in (0,1) \) and \( \varepsilon > 0 \), we say \( \mathcal{L} \) admits \((\varepsilon, \delta)\)-uniform convergence with \( N \) samples if for i.i.d. draws \( \tilde{X} = \{x_1, \ldots, x_N\} \sim \mathcal{D}^N \), it holds that

\[
\Pr_{\tilde{X}} \left[ \forall L \in \mathcal{L}, \left\| \frac{1}{N} \sum_{i=1}^N L(x_i) - \mathbb{E}_{x \sim \mathcal{D}} [L(x)] \right\| \leq \varepsilon \right] \geq 1 - \delta.
\]

If such a uniform bound over \( \mathcal{L} \) holds, we can bound the gap between the empirical and expected losses regardless of how sketching matrices are learned (e.g., manual or automatic).

The following pseudo- and fat shattering dimensions are fundamental notions of the complexity of function classes.

**Definition 1** (Pseudo- and fat shattering dimensions). Let \( \mathcal{L} \subseteq [0,1]^X \) be a class of functions. We say an input set \( \{x_1, \ldots, x_N\} \subseteq X \) is (pseudo) shatted by \( \mathcal{L} \) if there exist threshold values, \( t_1, \ldots, t_N \in \mathbb{R} \), satisfying the following condition: for every \( I \subseteq [N] \), there exists \( L \in \mathcal{L} \) such that

\[
i \in I \Rightarrow L(x_i) > t_i.
\]

For \( \gamma > 0 \), we say \( \{x_1, \ldots, x_N\} \subseteq X \) is \( \gamma \)-fat shatted by \( \mathcal{L} \) if the above condition holds with replacement of (1) by

\[
i \in I \Rightarrow L(x_i) > t_i + \gamma \quad \text{and} \quad i \notin I \Rightarrow L(x_i) < t_i - \gamma.
\]

The pseudo-dimension, \( \operatorname{pdim}(\mathcal{L}) \), and \( \gamma \)-fat shattering dimension, \( \operatorname{fatdim}_\gamma(\mathcal{L}) \), are the maximum size of a set that is pseudo and \( \gamma \)-fat shattered, respectively, by \( \mathcal{L} \).

It is well-known that \( N = \Omega(\varepsilon^{-2} \cdot (\operatorname{pdim}(\mathcal{L}) + \log \delta^{-1})) \) samples are sufficient for ensuring \((\varepsilon, \delta)\)-uniform convergence, and a similar guarantee holds if \( \operatorname{fatdim}_\gamma(\mathcal{L}) \) with \( \gamma = \Omega(\varepsilon) \) is bounded. We refer the reader to (Anthony and Bartlett, 1999, Theorems 19.1 and 19.2) for details.

2.2 Low-Rank Approximation

Let \( A \in \mathbb{R}^{n \times d} \) be an input matrix with \( n \geq d \). We assume \( \operatorname{rank}(A) > 0 \) and \( \|A\|_F^2 = 1 \) by normalization. For \( k \in [d] \),
Algorithm 1 SCW$_k(S, A)$

1: Compute $SA$
2: if $SA$ is a zero matrix:
3: return an $n \times d$ zero matrix
4: $U, \Sigma, V \leftarrow \text{SVD}(SA)$ \quad $\triangleright SA = U\Sigma V^T$
5: Compute $AV$
6: return $[AV]_{\|V^T}$

we consider computing a rank-$k$ approximation of $A$. Let $[A]_k \in \mathbb{R}^{n \times d}$ denote an optimal rank-$k$ approximation, i.e.,

$[A]_k \in \text{argmin}\{\|A - X\|_F^2 \mid X \in \mathbb{R}^{n \times d}, \text{rank}(X) = k\}$.

Although we can compute $[A]_k$ with SVD in $O(nd^2)$ time (Golub and Van Loan, 2013, Section 8.6.3), this approach is time and space consuming when $A$ is huge.

Algorithm 1 presents an efficient LRA algorithm with a sketching matrix $S \in \mathbb{R}^{n \times n}$ (Sarlos, 2006; Clarkson and Woodruff, 2009, 2017), which is called the SCW algorithm after the authors’ acronyms. Algorithm 1 is more efficient than computing $[A]_k$ if we set the sketching dimension, $m$, to a much smaller value than $d$, whereas we need $m \geq k$ to get a rank-$k$ approximation. Let SCW$_k(S, A)$ denote the output of Algorithm 1 with a sketching matrix $S$ and an input matrix $A$. It is known that for $\alpha > 0$, sketching matrices with $m = \Omega(k/\alpha)$ drawn from an appropriate distribution satisfy $\|A - \text{SCW}_k(S, A)\|_F \leq (1 + \alpha)\|A - [A]_k\|_F$ with high probability (e.g., (Woodruff, 2014, Section 4.1)).

Indyk et al. (2019) showed that machine-learned sketching matrices often enable more accurate LRA than random ones in practice. Given a training dataset $A_{\text{train}} \subseteq \mathbb{R}^{n \times d}$ of input matrices, they proposed to learn $S$ by minimizing the empirical risk $\frac{1}{|A_{\text{train}}|} \sum_{A \in A_{\text{train}}} \|A - \text{SCW}_k(S, A)\|_F^2$. Specifically, they learned sparse $S$ with the stochastic gradient descent method (SGD) by regarding non-zeros in $S$ at fixed positions as tunable parameters (where the sparsity of $S$ makes SCW$_k$ efficient). Later, researchers further studied learning-based LRA methods (Liu et al., 2020; Ailon et al., 2021; Indyk et al., 2021), which we will overview in Section 5.1.

2.3 Overview of (Bartlett et al., 2022)

Bartlett et al. (2022) formally studied learning-based LRA as a statistical learning problem. Let $A \subseteq \mathbb{R}^{n \times d}$ be a class of input matrices and $\mathcal{S} \subseteq \mathbb{R}^{m \times n}$ a class of sketching matrices, where every $S \in \mathcal{S}$ has up to $s$ non-zeros in each column and the sparsity pattern is identical for all $S \in \mathcal{S}$. Define a loss function $L : \mathcal{S} \times A \rightarrow [0, 1]^2$ based on SCW$_k$ as

$L(S, A) = \|A - \text{SCW}_k(S, A)\|_F^2$. \hspace{1cm} (2)

Let $\mathcal{L} = \{L(S, \cdot)\}_{S \in \mathcal{S}} \subseteq [0, 1]^4$ be the class of loss functions, where each $L(S, \cdot) \in \mathcal{L}$ is specified by $ns$ tunable parameters (non-zeros of $S$) and measures the approximation error of SCW$_k(S, \cdot)$. The authors presented the following $O(ns\cdot m + k\log(d/k) + \log(1/\varepsilon))$.

Theorem 1 (Bartlett et al. (2022, Theorem 2.2)). For sufficiently small $\varepsilon > 0$, the $\varepsilon$-fat shattering dimension of $\mathcal{L}$ is bounded as

$fatdim_\varepsilon(\mathcal{L}) = O(ns \cdot (m + k \log(d/k) + \log(1/\varepsilon)))$.

Intuitively, we can bound fatdim$_\varepsilon(\mathcal{L})$ by assessing the complexity of computational procedures for evaluating $L(S, A)$. In the LRA setting, however, directly bounding fatdim$_\varepsilon(\mathcal{L})$ is not easy since SCW$_k$ makes black-box use of SVD. The authors have overcome this difficulty by considering a class $\hat{\mathcal{L}}_\varepsilon$ of appropriate proxy loss functions, which we can evaluate with relatively simple computational procedures, and by bounding its pseudo-dimension, $pdim(\hat{\mathcal{L}}_\varepsilon)$. As in the following definition, each $\hat{L}_\varepsilon(S, \cdot) \in \hat{\mathcal{L}}_\varepsilon$ is evaluated with a power-method-based procedure so that $\hat{L}_\varepsilon(S, A)$ gives a sufficiently accurate approximation of $L(S, A)$.

Definition 2 (Proxy loss). For any $A \in \mathcal{A}$, $S \in \mathcal{S}$, and $\varepsilon > 0$, the proxy loss $\hat{L}_\varepsilon(S, A)$ is computed as follows:

1. Compute $B = A(SA)^{\dagger}(SA)$.
2. For all possible $P \in \mathbb{R}^{d \times k}$, compute $Z_i = (BB^\top)^{\dagger}BP_i$, where $q = O(\varepsilon^{-1}\log(d/\varepsilon))$.
3. Choose $Z = Z_i$ that minimizes $\|B - ZZ^\top B\|_F^2$.
4. $\hat{L}_\varepsilon(S, A) = \|A - ZZ^\top B\|_F^2$.

Given the class $\mathcal{S}$ of sketching matrices, the class of proxy loss functions is defined as $\hat{\mathcal{L}}_\varepsilon = \{\hat{L}_\varepsilon(S, \cdot)\}_{S \in \mathcal{S}}$.

As discussed in (Bartlett et al., 2022, Section 5.3), it holds that fatdim$_\varepsilon(\mathcal{L}) \leq pdim(\hat{\mathcal{L}}_\varepsilon)$. Therefore, an upper bound on $pdim(\hat{\mathcal{L}}_\varepsilon)$ immediately implies that on fatdim$_\varepsilon(\mathcal{L})$.

A benefit of considering $\hat{\mathcal{L}}_\varepsilon$ is that analyzing its complexity is easier than $\mathcal{L}$. The authors upper bounded $pdim(\hat{\mathcal{L}}_\varepsilon)$ by modeling the computational procedure of $\hat{L}_\varepsilon$ as a Goldberg–Jerrum algorithm (Goldberg and Jerrum, 1995).3

Definition 3 (Goldberg–Jerrum algorithm). A GJ algorithm $\Gamma$ takes real values as input, and its procedure is represented by a binary tree with the following two types of nodes:

• Computation node that executes an arithmetic operation $v'' = v \odot v'$, where $\odot \in \{+, -, \times, \div\}$.

• Branch node with an out-degree of 2, where branching is specified by the evaluation of a condition of the form $v \geq 0$ ($v \leq 0$) or $v = 0$.

3Such a notion is often called the algorithmic computation tree. Still, we here call it a GJ algorithm to be consistent with (Bartlett et al., 2022). Although their original definition does not contain the equality condition in branch nodes, dealing with equalities is easy due to (Goldberg and Jerrum, 1995, Corollary 2.1).
In both cases, \( \nu \) and \( \nu' \) are either inputs or values computed at ancestor nodes. Once input values are given, \( \Gamma \) proceeds along a root-leaf path on the tree and sequentially performs operations specified by nodes on the path.

Then, they defined two notions, the degree and predicate complexity, to measure the complexity of GJ algorithms.

**Definition 4** (Degree and predicate complexity). The degree of a GJ algorithm is the maximum degree of any rational function of input variables it computes. The predicate complexity of a GJ algorithm is the number of distinct rational functions that appear at its branch nodes. If a GJ algorithm has the degree and predicate complexity of at most \( \Delta \) and \( p \), respectively, we call it a \((\Delta, p)\)-GJ algorithm.

The following theorem says that if we can check whether a loss function value exceeds a threshold value or not using a \((\Delta, p)\)-GJ algorithm with small \( \Delta \) and \( p \), the class of such loss functions has a small pseudo-dimension.

**Theorem 2** (Bartlett et al., 2022, Theorem 3.3). Let \( X \) be an input domain and \( \mathcal{L} = \{ L_{\rho} : X \rightarrow \mathbb{R} \mid \rho \in \mathbb{R}^{\nu} \} \) a class of functions parameterized by \( \rho \in \mathbb{R}^{\nu} \). Assume that for every \( x \in X \) and \( t \in \mathbb{R} \), there is a \((\Delta, p)\)-GJ algorithm \( \Gamma_{x,t} \) that takes \( \rho \in \mathbb{R}^{\nu} \) as input and returns “true” if \( L_{\rho}(x) > t \) and “false” otherwise. Then, it holds that

\[
\text{pdim}(\mathcal{L}) = O(\nu \log(p\Delta)).
\]

The authors proved that for any \( A \in \mathcal{A} \) and \( t \in \mathbb{R} \), whether \( L_{\rho}(A, t) > 0 \) or not can be checked by a \((\Delta, p)\)-GJ algorithm \( \Gamma_{A,t} \) with \( \Delta = O(m k e^{-1} \log(d/\varepsilon)) \) and

\[
p = 2^{m \cdot 2O(k)} \cdot (d/k)^{3k},
\]

where input variables are \( ns \) non-zeros of \( S \), i.e., \( \nu = ns \). Therefore, Theorem 2 implies

\[
\text{pdim}(\mathcal{L}_{\nu}) = O(n s \cdot (m + k \log(d/k) + \log(1/\varepsilon))).
\]

The same bound applies to \( \text{fatdim}_{n}(\mathcal{L}) \leq \text{pdim}(\mathcal{L}_{\nu}) \), obtaining Theorem 1. They also gave an \( O(n s \nu) \) lower bound on \( \text{fatdim}_{n}(\mathcal{L}) \); hence it is tight up to an \( O(m) \) factor.

### 2.4 Warren’s Theorem

Warren’s theorem (Warren, 1968) is a useful tool to evaluate the complexity of a class of polynomials. The following extended version that allows the sign to be zero is presented in (Goldberg and Jerrum, 1995, Corollary 2.1).

**Theorem 3** (Warren’s theorem). Let \( \{ f_1, \ldots, f_N \} \) be a set of \( N \) polynomials of degree at most \( \Delta \) in \( \nu \) real variables \( \rho \in \mathbb{R}^{\nu} \). If \( N \geq \nu \), there are at most \((8e N \Delta / \nu)^{\nu}) \) distinct tuples of \( (\text{sign}(f_1(\rho)), \ldots, \text{sign}(f_N(\rho))) \in \{-1, 0, +1\}^N \).

This theorem is a key to proving Theorem 2, and we will also use it in Section 4. To familiarize ourselves with the theorem, we give a proof sketch of Theorem 2. From the statement assumption in Theorem 2, whether \( L_{\rho}(x) > t \) is determined by sign patterns of \( p \) polynomials of degree at most \( \Delta \) in \( \rho \in \mathbb{R}^{\nu} \) that appear at the branch nodes of the GJ algorithm, \( \Gamma_{x,t} \). Thus, when \( x_1, \ldots, x_N \in X \) and \( t_1, \ldots, t_N \in \mathbb{R} \) are given, the number of distinct outcomes (or tuples of \( N \) Booleans) of GJ algorithms \( \Gamma_{x_1,t_1}, \ldots, \Gamma_{x_N,t_N} \), which take common \( \rho \) as input, is bounded by the number of all possible sign patterns of \( Np \) polynomials of degree at most \( \Delta \) in \( \rho \).

From Warren’s theorem, the number of such sign patterns is at most \((8e N \Delta / \nu)^{\nu}) \), which must be at least \( 2^{N} \) to shatter \( x_1, \ldots, x_N \). The largest \( N \) with \((8e N \Delta / \nu)^{\nu}) \geq 2^{N} \) gives the \( O(\nu \log(p\Delta)) \) bound on \( \text{pdim}(\mathcal{L}) \), as in Theorem 2.

### 3 IMPROVED UPPER BOUND

We obtain an \( \tilde{O}(n s k) \) bound on \( \text{fatdim}_{n}(\mathcal{L}) \) by replacing the \( O(m) \) factor in (4) with \( O(\log m) \). To this end, we reduce the \( 2^m \) factor in the predicate complexity (3) to \( m \).

Note that, although concatenating random matrices with \( m = \tilde{O}(k/\alpha) \) rows guarantees the \((1 + \alpha)\)-approximation as mentioned in Section 2.2 (known as safeguard guarantees), our improvement is not meaningless since \( m \) can be much larger than \( k \). For example, even if we admit errors of \( SCW_k \) relative to \( |A|_k \) to the magnitude of \( \alpha \approx \varepsilon \), \( m = \tilde{O}(k/\alpha) \approx \tilde{O}(k/\varepsilon) \) does not imply \( m = \tilde{O}(k \log(d/k) + \log(1/\varepsilon)) \), hence \( O(n s k) \) can be significantly smaller than \( O(n s m) \).

#### 3.1 Previous Approach

We first explain where the \( 2^m \) factor comes from in (Bartlett et al., 2022). By carefully expanding the proof of (Bartlett et al., 2022, Lemma 5.6), one can confirm that it is caused by Step 1 in Definition 2, where a GJ algorithm computes \( Z \) with \( m \) rows (Bartlett et al., 2022, Lemma 5.2). We below describe their GJ algorithm for later convenience. In what follows, let \( I_r \) denote the \( r \times r \) identity matrix for any \( r \in \mathbb{Z}_{>0} \).

An essential tool for obtaining the GJ algorithm is the matrix inversion formula by the Cayley–Hamilton theorem.

**Proposition 1.** Let \( M \) be an \( r \times r \) real matrix and

\[
\det(\lambda I_r - M) = \lambda^r + c_1 \lambda^{r-1} + \cdots + c_r
\]

the characteristic polynomial of \( M \). If \( M \) is invertible, we have \( c_r = (-1)^r \det(M) \neq 0 \) and

\[
M^{-1} = -\frac{1}{c_r} \cdot (M^{r-1} + c_1 \cdot M^{r-2} + \cdots + c_{r-1} \cdot I_r).
\]

Let \( Z \) be an input matrix with \( m \) rows of rank \( r \leq m \). Their GJ algorithm computes \( Z^\dagger Z \) as follows. It first finds a matrix \( Y \) with \( r \) linearly independent rows selected from the rows of \( Z \). Since \( Y \) spans the row space of \( Z \), it holds

\( ^3 \text{Bartlett et al. (2022) alternatively used a recursive formula of (Csanky, 1976). This difference does not affect the conclusion.} \)
We present an algorithm that computes this using Proposition 1 with \( M = YY^\top \). Note that we have
\[
c_i = (-1)^{i} \sum_{S \subseteq [r]} \det(M[S]) \quad \text{for } i = 1, \ldots, r,
\]
where \( M[S] \) is the principal minor of \( M \) with indices \( S \subseteq [r] \); hence, if we take entries of \( M \) to be variables, \( c_1, \ldots, c_r \) are polynomials of degree at most \( r \). Thus, regarding entries of \( Z \) as variables, every rational function that appears in the above procedure has a degree of \( O(m) \).

What remains to be discussed is how to find \( Z^\top Z \); their algorithm computes this using Proposition 1 with \( M = YY^\top \). Note that we have
\[
c_i = (-1)^{i} \sum_{S \subseteq [r]} \det(M[S]) \quad \text{for } i = 1, \ldots, r,
\]
where \( M[S] \) is the principal minor of \( M \) with indices \( S \subseteq [r] \); hence, if we take entries of \( M \) to be variables, \( c_1, \ldots, c_r \) are polynomials of degree at most \( r \). Thus, regarding entries of \( Z \) as variables, every rational function that appears in the above procedure has a degree of \( O(m) \).

By performing Step 1 in Definition 2 with our GJ algorithm, we can replace the \( O(m) \) factor in the upper bound (4) with \( O(\log m) \), thus improving Theorem 1 as follows.

**Proposition 3.** For sufficiently small \( \varepsilon > 0 \), the \( \varepsilon \)-fat shattering dimension of \( \mathcal{L} \) is bounded as
\[
\text{fatdim}_\varepsilon(\mathcal{L}) = O(ns \cdot (\log m + k \log(d/k) + \log(1/\varepsilon))).
\]

### 3.3 Application to the Nyström Method

We briefly digress to demonstrate the usefulness of our GJ algorithm (Lemma 1). We here consider the classical Nyström method (Nyström, 1930). The method takes a positive semidefinite matrix \( A \in \mathbb{R}^{n \times n} \) as input and computes its rank-\( r \) approximation as \( AS(AS^\top AS)^{-1}(AS)^\top \), where \( S \in \mathbb{R}^{n \times r} \) is a sketching matrix. Unlike the SCW algorithm (Algorithm 1), it does not involve SVD, hence more efficient. Thus, it is a popular choice when handling large Laplacian and kernel matrices (Gittens and Mahoney, 2016).

As with learning-based LRA methods discussed so far, we can naturally combine the Nyström method with learning of sketching matrices. Specifically, defining a loss function as
\[
L(S, A) = \|A - AS(AS^\top AS)^{-1}(AS)^\top\|_F^2,
\]
we can learn high-performing sketching matrices from past data of \( A \) by minimizing the empirical risk. When it comes to generalization guarantees, we are interested in the pseudo-dimension of \( \mathcal{L} = \{L(S, \cdot)\}_{S \in S} \) with \( L \) defined as in (5), where we let \( S \subseteq \mathbb{R}^{n \times r} \) be a class of sketching matrices with \( \nu \) non-zeros at fixed positions.

We analyze the pseudo-dimension of \( \mathcal{L} \) by modeling the computational procedure of \( L(S, A) \) defined in (5) as a GJ algorithm. We first compute \((S^\top AS)^\top\) with our GJ algorithm (Lemma 1), whose degree and predicate complexity are \( O(r) \) and \( r \), respectively, where entries of \( S \) are variables. Other operations for computing \( L(S, A) \) require no branch nodes, and the degree remains \( O(r) \). Consequently, we can compute \( L(S, A) \) with an \( O(r) \)-GJ algorithm, and thus Theorem 2 implies the following bound on \( \text{pdim}(\mathcal{L}) \).

**Proposition 4.** For the class of \( \mathcal{L} \) of loss functions (5), each of which is parameterized by an \( n \times r \) sketching matrix \( S \in S \) with \( \nu \) non-zeros at fixed positions, it holds that
\[
\text{pdim}(\mathcal{L}) = O(\nu \log r).
\]

We can also deal with changeable sparsity patterns by using Theorem 4, which we will show in Section 4. In this case, it will immediately follow that \( \text{pdim}(\mathcal{L}) = O(\nu r \log r) \).

Note that if we compute \((S^\top AS)^\top\) with the previous GJ algorithm described in Section 3.1, its predicate complexity is \( 2^r \), resulting in \( \text{pdim}(\mathcal{L}) = O(\nu r \log r) \). This example suggests that our GJ algorithm can yield much better generalization bounds for classes of functions involving pseudo-inverse computation.
4 LEARNING SPARSITY PATTERNS

This section studies generalization bounds when sparsity patterns of sketching matrices can change. We show that even if the class $S$ of sketching matrices contains all $m \times n$ matrices with $ns$ non-zeros, the fat shattering dimension of $\mathcal{L} = \{ L(S, \cdot) \}_{S \in S}$ increases only by $O(ns \log n)$.

4.1 General Result

To deal with changeable sparsity patterns, we first present an extended version of Theorem 2.

**Theorem 4.** Let $X$ be an input domain and $\mathcal{L} \subseteq \mathbb{R}^X$ a class of functions with $\ell$ parameters $\rho \in \mathbb{R}^\ell$ that is $\nu$-sparse, i.e.,

$$\mathcal{L} = \{ L_\rho : X \rightarrow \mathbb{R} \mid \rho \in \mathbb{R}^\ell, |\text{supp}(\rho)| \leq \nu \}.$$

Assume that for every $x \in X$ and $t \in \mathbb{R}$, there is a $(\Delta, p)$-GJ algorithm, $\Gamma_{x,t}$, that takes a $\nu$-sparse variable vector $\rho \in \mathbb{R}^\ell$ as input and returns "true" if $L_\rho(x) > t$ and "false" otherwise. Then, we have

$$\text{pdim}(\mathcal{L}) = O(\nu \log((\ell p) \cdot \Delta)).$$

Compared with Theorem 2, there are $\ell$ $(\geq \nu)$ parameters, which are restricted to be $\nu$-sparse. If we naively use Theorem 2 without taking the sparsity into account, the pseudo-dimension bound turns out $O(\ell)$, even though every $L_\rho$ has only $\nu$ tunable non-zero parameters. Our Theorem 4 provides a refined bound that grows only logarithmically with $\ell$ and keeps the linear dependence on $\nu$.

The following proof idea comes from a PAC approach to one-bit compressed sensing (Ahsen and Vidyasagar, 2019), but how to use the idea is significantly different; indeed, the previous study does not combine it with Warren’s theorem.

**Proof of Theorem 4.** The proof proceeds similarly to that of (Bartlett et al., 2022, Theorem 3.3) (sketched in Section 2.4), but we must take changeable sparsity patterns into account.

We arbitrarily fix $N$ pairs, $(x_1, t_1), \ldots, (x_N, t_N)$, of an input and a threshold value. We upper bound the number of all possible tuples of $N$ Booleans (or outcomes) returned by the $N$ GJ algorithms, $\Gamma_{x_1,t_1}, \ldots, \Gamma_{x_N,t_N}$, whose input variable $\rho \in \mathbb{R}^\ell$ is any $\nu$-sparse vector. By the definition of the pseudo-dimension (see Definition 1), we need at least $2^N$ outcomes to shatter $\{x_1, \ldots, x_N\}$, and thus the largest such $N$ gives an upper bound on $\text{pdim}(\mathcal{L})$.

First, we fix a sparsity pattern $J \subseteq [\ell]$ with $|J| = \nu$ and let

$$\mathcal{L}_J = \{ L_\rho : X \rightarrow \mathbb{R} \mid \rho \in \mathbb{R}^\ell, \text{supp}(\rho) \subseteq J \}.$$

Note that we have $\mathcal{L} = \bigcup_{J \subseteq [\ell], |J| = \nu} \mathcal{L}_J$. From the statement assumption, there is a $(\Delta, \rho)$-GJ algorithm $\Gamma_{x,t}$ that can check whether $L_\rho(x) > t$ or not. That is, for any $(x, t)$, whether $L_\rho(x) > t$ or not is determined by sign patterns of $p$ polynomials of degree at most $\Delta$ in $\rho \in \mathbb{R}^\ell$. Moreover, since $\text{supp}(\rho) \subseteq J$, $\Gamma_{x,t}$ takes up to $\nu$ variables as input. Thus, once $J$ is fixed, outcomes of $\Gamma_{x_1,t_1}, \ldots, \Gamma_{x_N, t_N}$ are determined by sign patterns of $Np$ polynomials of degree at most $\Delta$ in $\nu$ variables. The number of such sign patterns is at most $(8eNp\Delta/\nu)^\nu$ by Warren’s theorem (Theorem 3).

Next, we consider changing sparsity patterns. As discussed above, a fixed sparsity pattern $J$ yields up to $(8eNp\Delta/\nu)^\nu$ outcomes of $\Gamma_{x_1,t_1}, \ldots, \Gamma_{x_N,t_N}$. If we feed $\rho \in \mathbb{R}^\ell$ with a new sparsity pattern $J'$ of size $\nu$ to $\Gamma_{x_1,t_1}, \ldots, \Gamma_{x_N,t_N}$, then $Np$ polynomials that appear in the GJ algorithms may exhibit up to $(8eNp\Delta/\nu)^\nu$ new sign patterns, which lead to at most that many new outcomes. Thus, when the sparsity pattern of $\rho$ can be any size-$\nu$ subset of $[\ell]$, the number of all possible outcomes of $\Gamma_{x_1,t_1}, \ldots, \Gamma_{x_N,t_N}$ is at most

"the number of sparsity patterns" $\times (8eNp\Delta/\nu)^\nu$.

Since there are up to $(\ell)^\nu$ sparsity patterns, the number of all possible outcomes of $\Gamma_{x_1,t_1}, \ldots, \Gamma_{x_N,t_N}$ is at most $(8eNp\Delta/\nu)^\nu$. In order for $\mathcal{L}$ to shatter $\{x_1, \ldots, x_N\}$,

$$2^N \leq (8eNp\Delta/\nu)^\nu \iff N \leq \nu \log_2(8eNp\Delta/\nu)$$

must hold. Since $\log_2 y \leq \frac{2}{3} y$ for $y > 0$, the right-hand side is bounded from above as

$$\nu \log_2(8eNp\Delta) + \nu \log_2(N/\nu) \leq \nu \log_2(8eNp\Delta) + \frac{2}{3} N.$$

Rearranging the terms, we obtain $N \leq 3\nu \log_2(8eNp\Delta)$, hence $\text{pdim}(\mathcal{L}) = O(\nu \log((\ell p) \cdot \Delta))$. 

4.2 Result on Learning-Based LRA

We now return to the LRA setting and discuss the pseudo-dimension bound for the case of changeable sparsity patterns. In this setting, we have $\ell = mn$ and $\nu = ns$ since every sketching matrix $S$ is of size $m \times n$ and has up to $ns$ non-zeros. Furthermore, from the discussion in Sections 2.3 and 3, for any input $A \in \mathcal{A}$ and threshold value $t \in \mathbb{R}$, we can check whether the proxy loss value, $\hat{L}_\epsilon(S, A)$, exceeds $t$ or not by using a $(\Delta, p)$-GJ algorithm with

$$\Delta = O(mk^{-1} \log(d/\epsilon))$$

and

$$p = m \cdot 2^{O(k)} \cdot (d/k)^3k.$$
Proposition 5. Let \( \mathcal{L} = \{ L(S, \cdot) \}_{S \in S} \) be the class of loss functions defined by (2) where \( S \) contains sketching matrices with any sparsity patterns of size \( ns \). For sufficiently small \( \varepsilon > 0 \), the \( \varepsilon \)-fat shattering dimension of \( \mathcal{L} \) is bounded as
\[
\text{fatdim}_\varepsilon(\mathcal{L}) = O(ns \cdot (\log(mn) + k \log(d/k) + \log(1/\varepsilon))).
\]

5 EXPERIMENTS

We confirm that learning sparsity patterns can improve the empirical accuracy of learning-based LRA methods. Note that the uniform bound discussed in Section 2.1 is agnostic to learning methods; therefore, we can use Proposition 5 to obtain generalization bounds for any methods to learn sparse sketching matrices.

5.1 Background and Learning Methods

Let us first overview existing methods for learning sketching matrices. Indyk et al. (2019) initiated the study of learning-based LRA, as mentioned in Section 2.2. Assuming fixed sparsity patterns, they learned sketching matrices by applying SGD to the SCW-based loss (2), where gradients are computed via backpropagation through differentiable SVD. Liu et al. (2020) enhanced the previous method by first learning sparsity patterns with a greedy algorithm and then learning non-zeros via SGD. A drawback of those two methods is that backpropagating through SVD is computationally expensive.\(^5\) Indyk et al. (2021) has overcome this issue by developing an efficient learning method based on a surrogate loss function. While their method again assumes fixed sparsity patterns, we can naturally extend it to changeable sparsity patterns, as detailed later. Another related work is (Ailon et al., 2021), which proposed to represent linear layers of neural networks as products of sparse matrices, like the butterfly networks. Although their idea is applicable to LRA, it requires sketching matrices with complicated structures; thus, we below do not consider it for simplicity.

Given the above background, a natural next direction is to extend the efficient method of (Indyk et al., 2021) to changeable sparsity patterns. In (Indyk et al., 2021), two kinds of methods are studied, one-shot and few-shot methods. We focus on the latter and present how to modify it to learn both positions and values of non-zeros. Their basic idea is to minimize the following surrogate loss instead of the SCW-based loss (2):
\[
\tilde{L}(S, A) = \| U_k^T S^T S U - I_0 \|_F^2,
\]
where \( U \in \mathbb{R}^{n \times k} \) is the column orthogonal matrix computed by SVD of \( A \) (assuming \( \text{rank}(A) = d \)), \( U_k \in \mathbb{R}^{n \times k} \) is the first \( k \) columns of \( U \) corresponding to the largest \( k \) singular values, and \( I_0 = [I_k, 0_{k,d-k}] \in \mathbb{R}^{k \times d} \) is a concatenation of the \( k \times k \) identity matrix and \( k \times d - k \) zeros. Unlike the SCW-loss, differentiating the surrogate loss, \( \tilde{L}(S, A) \), with respect to \( S \) does not require backpropagation through SVD, hence more efficient. Moreover, (Indyk et al., 2021, Theorem 2.2) ensures the consistency of the surrogate loss, i.e., \( \tilde{L}(S, A) \leq \varepsilon \) implies \( \| A - \text{SCW}_k(S, A) \|_F^2 \leq (1 + O(\varepsilon))\| A - [A]_k \|_F^2 \). By minimizing the empirical surrogate loss via SGD, they learned non-zeros of sketching matrices at fixed positions.

To learn both positions and values of non-zeros based on the above idea, we use the projected gradient descent method, or sometimes called iterative hard thresholding (IHT) in non-convex sparse optimization (Jain and Kar, 2017). The method works iteratively as with SGD. In each iteration, given an input matrix \( A \in \mathcal{A}_{\text{train}} \) in a training dataset, we update the sketching matrix as \( S \leftarrow \Pi_k(S - \eta \nabla L(S, A)) \), where \( \eta > 0 \) is a step size, \( \nabla L(S, A) \) is the gradient with respect to \( S \), and \( \Pi_k \) is a projection operator that preserves the largest \( s \) elements in absolute value for each column and set the others to zero. In the following experiments, we refer to this method, which learns positions of non-zeros, as Learn and compare it with two baselines: Fix and Dense. Fix is the method studied in (Indyk et al., 2021), which learns non-zeros at fixed positions via SGD. Dense learns values of all entries via SGD. Note that although Dense naturally attains the best accuracy among them, it results in dense sketching matrices, which cannot benefit from the efficiency of sparse matrix multiplication and cause longer runtime of SCW\(_k\) when deployed for future data.

5.2 Settings and Results

Experiments were conducted on a macOS machine with Apple M2 CPU and 24 GB RAM. We implemented the methods in Python 3.9.12 and used JAX 0.3.15 (Bradbury et al., 2018) to compute gradients. When performing SVD, we regarded singular values smaller than \( 10^{-8} \) as zero.

Let \( n = 100 \), \( d = 50 \), \( m = 10 \), and \( k = 5 \). We made a rank-\( k \) matrix \( A_{\text{true}} \in \mathbb{R}^{n \times d} \) by multiplying \( n \times k \) and \( k \times d \) matrices whose entries were drawn from the uniform distributions over \([0, 1]\). We then let \( A = A_{\text{true}} + 0.1 \times A_{\text{noise}} \), where entries of \( A_{\text{noise}} \) were drawn from the standard normal distributions, and normalized \( A \) so that \( \|A\|_F = 1 \) holds. By drawing 300 noise terms independently, we created a dataset of 300 input matrices \( A \). We split them into training and test datasets of sizes 200 and 100, respectively. We made 30 random training/test splits to calculate the average and standard deviation over the 30 random trials.

We learn sketching matrices \( S \in \mathbb{R}^{m \times n} \) by minimizing the empirical surrogate loss (6) on a training dataset. Fix and Learn learn \( S \) with \( s = 1 \), 3, or 5 non-zeros in each column; since \( m = 10 \), the \( s \) values mean that 10%, 30%, or 50% of entries can be non-zero, respectively. Initial sketching matrices were obtained by setting random \( s \) entries in each column to \(-1 \) or \(+1\) with probability 0.5, respectively, and

\(^5\) Li et al. (2023) updated (Liu et al., 2020) and independently addressed this drawback.
As with the training SCW-based loss values (Figure 1), the gap between \( \text{Fix} \) and \( \text{Learn} \) was evident with \( s = 1 \) and 3, while both achieved as small losses as \( \text{Dense} \) when \( s = 5 \).

To conclude, \( \text{Learn} \) achieved smaller SCW-based loss values than \( \text{Fix} \) particularly when \( s \) was small, suggesting that learning sparsity patterns enables more accurate learning-based LRA when we need to learn highly sparse sketching matrices for the sake of the efficiency of \( \text{SCW}_k \).

Figure 1 shows curves of surrogate (6) and SCW-based (2) loss values in the training phase. As \( s \) increased, the performances of \( \text{Fix} \) and \( \text{Learn} \) became closer to that of \( \text{Dense} \). Regarding the surrogate loss, \( \text{Learn} \) achieved smaller values than \( \text{Fix} \), implying that \( \text{Learn} \) could go beyond local optima into which \( \text{Fix} \) fell. As for the SCW-based loss, \( \text{Learn} \) slightly outperformed \( \text{Fix} \) for \( s = 1 \) and 3, and both achieved almost as small values as \( \text{Dense} \) when \( s = 5 \).

Figure 2 shows the SCW-based loss values on test datasets. As with the training SCW-based loss values (Figure 1), the gap between \( \text{Fix} \) and \( \text{Learn} \) was evident with \( s = 1 \) and 3, while both achieved as small losses as \( \text{Dense} \) with \( s = 5 \).

As for training times, \( \text{Learn} \) took about 8% longer than \( \text{Fix} \), although our main focus is accuracy and the implementations are not intended to be fast.

### 6 CONCLUSION AND DISCUSSION

Building on (Bartlett et al., 2022), we have studied generalization bounds for learning-based LRA. We have improved their \( \tilde{O}(nsm) \) bound on the fat shattering dimension to \( \tilde{O}(nsk) \) by developing an \( (O(m), m) \)-GJ algorithm that computes a pseudo-inverse of a matrix with \( m \) rows. We have also demonstrated its usefulness by applying it to the learning-based Nyström method. Then, we have shown that learning both positions and values of non-zeros of sketching matrices increases the fat-shattering-dimension bound only by \( O(ns \log n) \). Experiments have confirmed that the efficient learning method of (Indyk et al., 2021) can achieve higher empirical accuracy with changeable sparsity patterns.

A notable open problem is to close the \( \tilde{O}(k) \) gap between the \( \tilde{O}(nsm) \) upper and \( \Omega(ns) \) lower bounds. Note that only applying our GJ algorithm to Step 3 in Definition 2 does not leave out the \( \tilde{O}(k) \) factor; a more essential problem lies in Step 2, where we must avoid using exponentially many \( P_i \) in \( k \) to remove the \( \tilde{O}(k) \) factor. When it comes to improving the \( \Omega(ns) \) lower bound, we need to shatter more instances than \( \Omega(ns) \), where \( ns \) is the number of tunable parameters. Although obtaining a greater lower bound than the number of tunable parameters is typically challenging, such lower bounds have been obtained for neural networks using the bit extraction technique (Bartlett et al., 1998). We expect that a similar idea would help obtain a tighter lower bound.
Acknowledgments

This work was supported by JST ERATO Grant Number JPMJER1903 and JSPS KAKENHI Grant Number JP22K17853.

References


