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 $\tilde{\mathrm{O}}(nsm)$ bound on the *fat shattering dimension* ($\tilde{\mathrm{O}}$ hides logarithmic factors). We build on their work and make two contributions.

(1) We present a better $\tilde{\mathrm{O}}(nsk)$ 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 $\mathrm{O}(ns \log n)$. 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 $\tilde{\mathrm{O}}(nsm)$[1] 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 $\tilde{\mathrm{O}}(m)$ 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 $\tilde{\mathrm{O}}(nsm)$ upper bound by replacing the $\mathrm{O}(m)$ factor with $\mathrm{O}(\log m)$, which leads to a better $\tilde{\mathrm{O}}(nsk)$ 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 $m = \mathrm{O}(k)$. 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*

Proceedings of the 26th International Conference on Artificial Intelligence and Statistics (AISTATS) 2023, Valencia, Spain. PMLR: Volume 206.

[1] We use $\tilde{\mathrm{O}}$ 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 $\tilde{\mathrm{O}}(nsk)$ upper bound on the fat shattering dimension increases only by $\mathrm{O}(ns \log n)$, despite the presence of exponentially many possible sparsity patterns in ns. Hence, the bound remains $\tilde{\mathrm{O}}(nsk)$ (ignoring $\mathrm{O}(\log n)$) 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 $[n] = \{1, \dots, n\}$. Let $\mathrm{sign}(\cdot)$ 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 rank, tr, and $\det$ denote the rank, trace, and determinant. Let $\|A\|_F = \sqrt{\mathrm{tr}(A^\top A)}$ denote the Frobenius norm of a matrix A. The Moore–Penrose pseudo-inverse of a matrix A is denoted by $A^\dagger$. SVD refers to the compact singular value decomposition, i.e., for $A \in \mathbb{R}^{n \times d}$ with $\mathrm{rank}(A) = r$, 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 $\mathrm{supp}(x) \subseteq [n]$ 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 $\mathrm{supp}(x) \subseteq J$.

2.1 Learning Theory

Let $\mathcal{X}$ be a domain of inputs, $\mathcal{D}$ a distribution over $\mathcal{X}$, and $\mathcal{L} \subseteq [0,1]^{\mathcal{X}}$ a class of loss functions. In our case, $\mathcal{X}$ 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 (ε, δ)-*uniform convergence* with N samples if for i.i.d. draws $\tilde{X} = \{x_1, \dots, x_N\} \sim \mathcal{D}^N$, it holds that

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

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

$$i \in I \Leftrightarrow L(x_i) > t_i. \tag{1}$$

For $\gamma > 0$, we say $\{x_1, \dots, x_N\} \subseteq \mathcal{X}$ is γ-*fat shattered* 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*, $\mathrm{pdim}(\mathcal{L})$, and γ-*fat shattering dimension*, $\mathrm{fatdim}_\gamma(\mathcal{L})$, are the maximum size of a set that is pseudo and γ-fat shattered, respectively, by $\mathcal{L}$.

It is well-known that $N = \Omega(\varepsilon^{-2} \cdot (\mathrm{pdim}(\mathcal{L}) + \log \delta^{-1}))$ samples are sufficient for ensuring (ε, δ)-uniform convergence, and a similar guarantee holds if $\mathrm{fatdim}_\gamma(\mathcal{L})$ 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 $n \ge d$. We assume $\mathrm{rank}(A) > 0$ and $\|A\|_F^2 = 1$ by normalization. For $k \in [d]$,

Algorithm 1 $\mathrm{SCW}_k(S, A)$

Compute SA
if SA is a zero matrix **:**
  return an $n \times d$ zero matrix
$U, \Sigma, V \leftarrow \mathrm{SVD}(SA)$ ▷ $SA = U\Sigma V^\top$
Compute AV
return $[AV]_k V^\top$

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.,

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

Although we can compute $[A]_k$ with SVD in $\mathrm{O}(nd^2)$ 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 $S \in \mathbb{R}^{m\times n}$ (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 $\mathrm{SCW}_k(S, A)$ 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 $m = \tilde{\Omega}(k/\alpha)$ drawn from an appropriate distribution satisfy $\|A - \mathrm{SCW}_k(S, A)\|_F \leq (1+\alpha)\|A - [A]_k\|_F$ 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 $\mathcal{A}_{\text{train}} \subseteq \mathbb{R}^{n\times d}$ of input matrices, they proposed to learn S by minimizing the empirical risk $\frac{1}{|\mathcal{A}_{\text{train}}|}\sum_{A\in\mathcal{A}_{\text{train}}} \|A - \mathrm{SCW}_k(S, A)\|_F^2$. 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 $\mathcal{A} \subseteq \mathbb{R}^{n\times d}$ 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 $L : \mathcal{S} \times \mathcal{A} \to [0, 1]$[2] based on SCW_k as

$$L(S, A) = \|A - \mathrm{SCW}_k(S, A)\|_F^2. \tag{2}$$

Let $\mathcal{L} = \{L(S, \cdot)\}_{S\in\mathcal{S}} \subseteq [0, 1]^{\mathcal{A}}$ 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 $\mathrm{SCW}_k(S, \cdot)$. The authors presented the following $\tilde{\mathrm{O}}(nsm)$ bound on the ε-fat shattering dimension of $\mathcal{L}$.

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

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

Intuitively, we can bound $\mathrm{fatdim}_\varepsilon(\mathcal{L})$ by assessing the complexity of computational procedures for evaluating $L(S, A)$. In the LRA setting, however, directly bounding $\mathrm{fatdim}_\varepsilon(\mathcal{L})$ 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, $\mathrm{pdim}(\hat{\mathcal{L}}_\varepsilon)$. 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 $P_i \in \mathbb{R}^{d\times k}$ ($i = 1, \ldots, \binom{d}{k}$) whose columns are k distinct standard vectors in $\mathbb{R}^d$, compute $Z_i = (BB^\top)^q BP_i$, where $q = \mathrm{O}(\varepsilon^{-1}\log(d/\varepsilon))$.
3. Choose $Z = Z_i$ that minimizes $\|B - Z_i Z_i^\dagger B\|_F^2$.
4. $\hat{L}_\varepsilon(S, A) = \|A - ZZ^\dagger B\|_F^2$.

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 $\mathrm{fatdim}_\varepsilon(\mathcal{L}) \leq \mathrm{pdim}(\hat{\mathcal{L}}_\varepsilon)$. Therefore, an upper bound on $\mathrm{pdim}(\hat{\mathcal{L}}_\varepsilon)$ immediately implies that on $\mathrm{fatdim}_\varepsilon(\mathcal{L})$.

A benefit of considering $\hat{\mathcal{L}}_\varepsilon$ is that analyzing its complexity is easier than $\mathcal{L}$. The authors upper bounded $\mathrm{pdim}(\hat{\mathcal{L}}_\varepsilon)$ 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 Γ 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 $v = 0$.

[2] $L(S, A)$ is at most $\|A\|_F^2 = 1$, as in (Bartlett et al., 2022).

[3] Such 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, v and v' are either inputs or values computed at ancestor nodes. Once input values are given, Γ 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 Δ and p, respectively, we call it a (Δ, 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 (Δ, p)-GJ algorithm with small Δ and p, the class of such loss functions has a small pseudo-dimension.

Theorem 2 (Bartlett et al. (2022, Theorem 3.3))**.** *Let $\mathcal{X}$ be an input domain and $\mathcal{L} = \{ L_\rho : \mathcal{X} \to \mathbb{R} \mid \rho \in \mathbb{R}^\nu \}$ a class of functions parameterized by $\rho \in \mathbb{R}^\nu$. Assume that for every $x \in \mathcal{X}$ and $t \in \mathbb{R}$, there is a (Δ, 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*

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

The authors proved that for any $A \in \mathcal{A}$ and $t \in \mathbb{R}$, whether $\hat{L}_\varepsilon(S, A) > t$ or not can be checked by a (Δ, p)-GJ algorithm $\Gamma_{A,t}$ with $\Delta = \mathrm{O}(mk\varepsilon^{-1} \log(d/\varepsilon))$ and

$$p = 2^m \cdot 2^{\mathrm{O}(k)} \cdot (d/k)^{3k}, \tag{3}$$

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

$$\mathrm{pdim}(\hat{\mathcal{L}}_\varepsilon) = \mathrm{O}(ns \cdot (m + k \log(d/k) + \log(1/\varepsilon))). \tag{4}$$

The same bound applies to $\mathrm{fatdim}_\varepsilon(\mathcal{L})$ ($\leq \mathrm{pdim}(\hat{\mathcal{L}}_\varepsilon)$), obtaining Theorem 1. They also gave an $\Omega(ns)$ lower bound on $\mathrm{fatdim}_\varepsilon(\mathcal{L})$; hence it is tight up to an $\tilde{\mathrm{O}}(m)$ 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 Δ in ν real variables $\rho \in \mathbb{R}^\nu$. If $N \geq \nu$, there are at most $(8\mathrm{e}N\Delta/\nu)^\nu$ distinct tuples of $(\mathrm{sign}(f_1(\rho)), \ldots, \mathrm{sign}(f_N(\rho))) \in \{-1, 0, +1\}^N$.*

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 Δ in $\rho \in \mathbb{R}^\nu$ that appear at the branch nodes of the GJ algorithm, $\Gamma_{x,t}$. Thus, when $x_1, \ldots, x_N \in \mathcal{X}$ 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 ρ as input, is bounded by the number of all possible sign patterns of Np polynomials of degree at most Δ in ρ. From Warren's theorem, the number of such sign patterns is at most $(8\mathrm{e}Np\Delta/\nu)^\nu$, which must be at least 2^N to shatter $x_1, \ldots, x_N$. The largest N with $(8\mathrm{e}Np\Delta/\nu)^\nu \geq 2^N$ gives the $\mathrm{O}(\nu \log(p\Delta))$ bound on $\mathrm{pdim}(\mathcal{L})$, as in Theorem 2.

3 IMPROVED UPPER BOUND

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

Note that, although concatenating random matrices with $m = \tilde{\mathrm{O}}(k/\alpha)$ 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 lager than k. For example, even if we admit errors of SCW_k relative to $[A]_k$ to the magnitude of $\alpha \approx \varepsilon$, $m = \tilde{\mathrm{O}}(k/\alpha) \simeq \tilde{\mathrm{O}}(k/\varepsilon)$ does not imply $m = \mathrm{O}(k \log(d/k) + \log(1/\varepsilon))$, hence $\tilde{\mathrm{O}}(nsk)$ can be significantly smaller than $\tilde{\mathrm{O}}(nsm)$.

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 $A(SA)^\dagger(SA)$. For this step, they used an $(\mathrm{O}(m), 2^m)$-GJ algorithm that computes $Z^\dagger Z$ for an input matrix 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.[4]

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

[4] Bartlett et al. (2022) alternatively used a recursive formula of (Csanky, 1976). This difference does not affect the conclusion.

$Z^\dagger Z = Y^\top (YY^\top)^{-1} Y$; their algorithm computes this using Proposition 1 with $M = YY^\top$. Note that we have

$$c_i = (-1)^i \sum_{S:|S|=i} \det M[S] \qquad \text{for } i = 1, \dots, r,$$

where $M[S]$ is the principal minor of M with indices $S \subseteq [r]$; hence, if we take entries of M to be variables, $c_1, \dots, c_r$ 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 $\mathrm{O}(m)$.

What remains to be discussed is how to find Y of full row rank. To achieve this, their GJ algorithm goes over the rows of Z and sequentially adds appropriate rows to Y in a greedy fashion. Whenever adding a new row, it checks whether the resulting Y has full row rank by examining whether $\det(YY^\top) \neq 0$ or not. This procedure involves polynomials of degree $\mathrm{O}(m)$, and the number of branch nodes is up to 2^m depending on which rows of Z are selected, resulting in the 2^m predicate complexity.

3.2 Our Result

We present an $(\mathrm{O}(m), m)$-GJ algorithm for computing $Z^\dagger$ (right-multiplying Z only increases the degree by one).

Lemma 1. *Let Z be an input matrix with m rows. There is an $(\mathrm{O}(m), m)$-GJ algorithm that computes $Z^\dagger$.*

Our key idea is to begin by determining $r = \mathrm{rank}(ZZ^\top)$ with m branch nodes, instead of branching to determine the choice of rows of Z. Once r is fixed, we can calculate $Z^\dagger$ without branching by the following formula.

Proposition 2 (Decell (1965, Theorem 3))**.** *Let Z be a matrix with m rows and $c_1, \dots, c_m$ the coefficients of the characteristic polynomial of $M = ZZ^\top \in \mathbb{R}^{m \times m}$, i.e.,*

$$\det(\lambda I_m - M) = \lambda^m + c_1 \lambda^{m-1} + \cdots + c_m.$$

If $r \geq 1$ is the largest index with $c_r \neq 0$, we have

$$Z^\dagger = -\frac{1}{c_r} \cdot Z^\top \left(M^{r-1} + c_1 \cdot M^{r-2} + \cdots + c_{r-1} \cdot I_m \right).$$

If $c_1 = \cdots = c_m = 0$, $Z^\dagger$ is a zero matrix.

By using this formula in lieu of Proposition 1, we can obtain an $(\mathrm{O}(m), m)$-GJ algorithm that computes $Z^\dagger Z$.

Proof of Lemma 1. We give a concrete GJ algorithm. Let $M = ZZ^\top$. First, we compute the coefficients $c_1, \dots, c_m$ of $\det(\lambda I_m - M)$, which are polynomials of degree $\mathrm{O}(m)$ in the entries of Z. Then, check whether $c_i \neq 0$ in decreasing order of i. Once we find $c_i \neq 0$, set $r = i$ as the largest index r with $c_r \neq 0$. Note that this requires only m branch nodes. If $c_m = \cdots = c_1 = 0$, let $Z^\dagger$ be a zero matrix. Otherwise, we compute $Z^\dagger$ as in Proposition 2. Every rational function in the above calculation has a degree of $\mathrm{O}(m)$ in Z. Thus, we obtain a desired $(\mathrm{O}(m), m)$-GJ algorithm. □

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

Proposition 3. *For sufficiently small $\varepsilon > 0$, the ε-fat shattering dimension of $\mathcal{L}$ is bounded as*

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

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 $AS(S^\top AS)^\dagger (AS)^\top$, 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

$$L(S, A) = \|A - AS(S^\top AS)^\dagger (AS)^\top\|_F^2, \tag{5}$$

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 $\mathcal{L} = \{L(S, \cdot)\}_{S \in \mathcal{S}}$ with L defined as in (5), where we let $\mathcal{S} \subseteq \mathbb{R}^{n \times r}$ be a class of sketching matrices with ν 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 $(S^\top AS)^\dagger$ with our GJ algorithm (Lemma 1), whose degree and predicate complexity are $\mathrm{O}(r)$ and r, respectively, where entries of S are variables. Other operations for computing $L(S, A)$ require no branch nodes, and the degree remains $\mathrm{O}(r)$. Consequently, we can compute $L(S, A)$ with an $(\mathrm{O}(r), r)$-GJ algorithm, and thus Theorem 2 implies the following bound on $\mathrm{pdim}(\mathcal{L})$.

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

$$\mathrm{pdim}(\mathcal{L}) = \mathrm{O}(\nu \log r).$$

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 $\mathrm{pdim}(\mathcal{L}) = \mathrm{O}(\nu \log(nr))$.

Note that if we compute $(S^\top AS)^\dagger$ with the previous GJ algorithm described in Section 3.1, its predicate complexity is 2^r, resulting in $\mathrm{pdim}(\mathcal{L}) = \mathrm{O}(\nu r \log r)$. Thus, 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 $\mathcal{S}$ of sketching matrices contains all $m \times n$ matrices with ns non-zeros, the fat shattering dimension of $\mathcal{L} = \{L(S, \cdot)\}_{S \in \mathcal{S}}$ increases only by $\mathrm{O}(ns \log n)$.

4.1 General Result

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

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

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

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

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

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

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, $(x_1, t_1), \dots, (x_N, t_N)$, 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, $\Gamma_{x_1,t_1}, \dots, \Gamma_{x_N,t_N}$, whose input variable $\rho \in \mathbb{R}^{\ell}$ is any ν-sparse vector. By the definition of the pseudo-dimension (see Definition 1), we need at least 2^N outcomes to shatter $\{x_1, \dots, x_N\}$, and thus the largest such N gives an upper bound on $\operatorname{pdim}(\mathcal{L})$.

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

$$\mathcal{L}_J = \left\{ L_\rho : \mathcal{X} \to \mathbb{R} \;\middle|\; \rho \in \mathbb{R}^{\ell},\ \operatorname{supp}(\rho) \subseteq J \right\}.$$

Note that we have $\mathcal{L} = \bigcup_{J \subseteq [\ell] : |J| = \nu} \mathcal{L}_J$. From the statement assumption, there is a (Δ, p)-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 Δ in $\rho \in \mathbb{R}^{\ell}$. Moreover, since $\operatorname{supp}(\rho) \subseteq J$, $\Gamma_{x,t}$ takes up to ν variables as input. Thus, once J is fixed, outcomes of $\Gamma_{x_1,t_1}, \dots, \Gamma_{x_N,t_N}$ are determined by sign patterns of Np polynomials of degree at most Δ in ν variables. The number of such sign patterns is at most $(8\mathrm{e}Np\Delta/\nu)^{\nu}$ by Warren's theorem (Theorem 3).

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

$$\text{"the number of sparsity patterns"} \times (8\mathrm{e}Np\Delta/\nu)^{\nu}.$$

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

$$2^N \le (8\mathrm{e}\ell Np\Delta/\nu)^{\nu} \Leftrightarrow N \le \nu \log_2(8\mathrm{e}\ell Np\Delta/\nu)$$

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

$$\nu \log_2(8\mathrm{e}\ell p\Delta) + \nu \log_2(N/\nu) \le \nu \log_2(8\mathrm{e}\ell p\Delta) + \frac{2}{3} N.$$

Rearranging the terms, we obtain $N \le 3\nu \log_2(8\mathrm{e}\ell p\Delta)$, hence $\operatorname{pdim}(\mathcal{L}) = \mathrm{O}(\nu \log(\ell p \Delta))$. □

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, $\hat{L}_\varepsilon(S, A)$, exceeds t or not by using a (Δ, p)-GJ algorithm with

$$\Delta = \mathrm{O}(mk\varepsilon^{-1} \log(d/\varepsilon)) \ \text{ and } \ p = m \cdot 2^{\mathrm{O}(k)} \cdot (d/k)^{3k}.$$

Thus, from Theorem 4, for the class $\hat{\mathcal{L}}_\varepsilon = \{\hat{L}_\varepsilon(S, \cdot)\}_{S \in \mathcal{S}}$ of proxy loss functions where $\mathcal{S}$ consists of sketching matrices with ns non-zeros at any positions, it holds that

$$\operatorname{pdim}(\hat{\mathcal{L}}_\varepsilon) = \mathrm{O}(ns \cdot (\log(mn) + k \log(d/k) + \log(1/\varepsilon))).$$

The right-hand side is larger than the bound in Proposition 3 only by $\mathrm{O}(ns \log n)$. Note that narrowing the class $\mathcal{S}$ only decreases $\operatorname{pdim}(\hat{\mathcal{L}}_\varepsilon)$; hence, the bound remains true when each $S \in \mathcal{S}$ is restricted to have s non-zeros in each column. Since we have $\operatorname{fatdim}_\varepsilon(\mathcal{L}) \le \operatorname{pdim}(\hat{\mathcal{L}}_\varepsilon)$ as discussed in Section 2.3, we obtain the following result.

Proposition 5. *Let $\mathcal{L} = \{L(S,\cdot)\}_{S\in\mathcal{S}}$ be the class of loss functions defined by (2) where $\mathcal{S}$ contains sketching matrices with any sparsity patterns of size ns. For sufficiently small $\varepsilon > 0$, the ε-fat shattering dimension of $\mathcal{L}$ is bounded as*

$$\mathrm{fatdim}_\varepsilon(\mathcal{L}) = \mathrm{O}(ns\cdot(\log(mn)+k\log(d/k)+\log(1/\varepsilon))).$$

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):

$$\tilde{L}(S,A) = \|U_k^\top S^\top SU - I_0\|_F^2, \tag{6}$$

where $U \in \mathbb{R}^{n\times d}$ is the column orthogonal matrix computed by SVD of A (assuming $\mathrm{rank}(A) = d$), $U_k \in \mathbb{R}^{n\times k}$ is the first k columns of U corresponding to the largest k singular values, and $I_0 = [I_k, \mathbf{0}_{k,d-k}] \in \mathbb{R}^{k\times d}$ 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., $\tilde{L}(S,A) \le \varepsilon$ implies $\|A - \mathrm{SCW}_k(S,A)\|_F^2 \le (1+\mathrm{O}(\varepsilon))\|A-[A]_k\|_F^2$. 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 $S \leftarrow \Pi_s(S - \eta\nabla\tilde{L}(S,A))$, where $\eta > 0$ is a step size, $\nabla\tilde{L}(S,A)$ is the gradient with respect to S, and Π_s 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_{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.

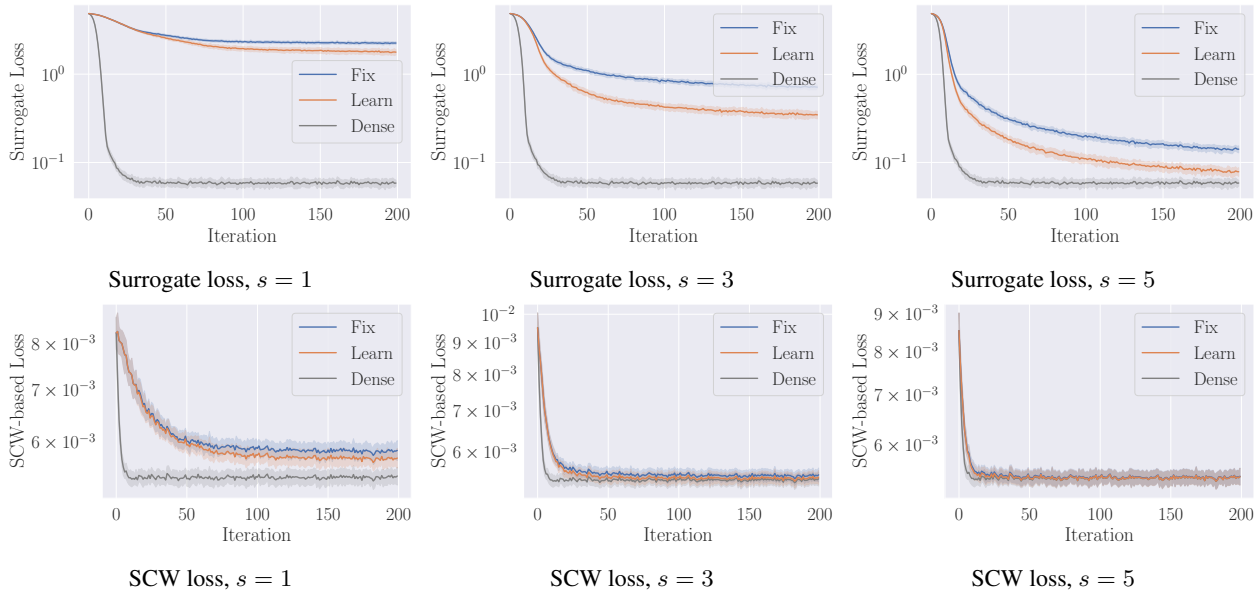

Figure 1: Surrogate and SCW-based loss values on training datasets. The x-axis indicates the number of iterations of SGD (for **Fix** and **Dense**) or stochastic IHT (for **Learn**). The error band indicates the standard deviation over the 30 random trials.

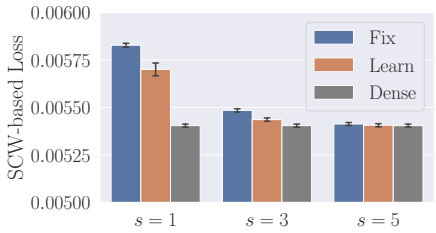

Figure 2: SCW-based loss values on test datasets. The error bar shows the standard deviation over the 30 random trials.

the others to zero; we then normalized it to satisfy $\|S\|_F = 1$ for numerical stability. We set the step size, η, to 0.1.

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

To conclude, **Learn** achieved smaller SCW-based loss values than **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 SCW_k.

As for training times, **Learn** took about 8% longer than **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 $\tilde{\mathrm{O}}(nsm)$ bound on the fat shattering dimension to $\tilde{\mathrm{O}}(nsk)$ by developing an $(\mathrm{O}(m), m)$-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 $\mathrm{O}(ns \log n)$. 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 $\tilde{\mathrm{O}}(k)$ gap between the $\tilde{\mathrm{O}}(nsk)$ upper and $\Omega(ns)$ lower bounds. Note that only applying our GJ algorithm to Step 3 in Definition 2 does not leave out the $\tilde{\mathrm{O}}(k)$ factor; a more essential problem lies in Step 2, where we must avoid using exponentially many P_i in k to remove the $\tilde{\mathrm{O}}(k)$ 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

M. E. Ahsen and M. Vidyasagar. An approach to one-bit compressed sensing based on probably approximately correct learning theory. *J. Mach. Learn. Res.*, 20(11): 1–23, 2019. ↰ p.6

N. Ailon, O. Leibovitch, and V. Nair. Sparse linear networks with a fixed butterfly structure: Theory and practice. In *Proceedings of the 37th Conference on Uncertainty in Artificial Intelligence (UAI 2021)*, volume 161, pages 1174–1184. PMLR, 2021. ↰ p.3, ↰ p.7

M. Anthony and P. L. Bartlett. *Neural Network Learning: Theoretical Foundations*. Cambridge University Press, 1999. ↰ p.2

M.-F. Balcan. Data-driven algorithm design. In *Beyond the Worst-Case Analysis of Algorithms*, pages 626–645. Cambridge University Press, 2021. ↰ p.2

M.-F. Balcan, T. Dick, T. Sandholm, and E. Vitercik. Learning to branch. In *Proceedings of the 35th International Conference on Machine Learning (ICML 2018)*, volume 80, pages 344–353. PMLR, 2018. ↰ p.2

M.-F. Balcan, T. Dick, and M. Lang. Learning to link. In *Proceedings of the International Conference on Learning Representations (ICLR 2020)*, 2020a. ↰ p.2

M.-F. Balcan, T. Sandholm, and E. Vitercik. Refined bounds for algorithm configuration: The knife-edge of dual class approximability. In *Proceedings of the 37th International Conference on Machine Learning (ICML 2020)*, volume 119, pages 580–590. PMLR, 2020b. ↰ p.2

M.-F. Balcan, D. DeBlasio, T. Dick, C. Kingsford, T. Sandholm, and E. Vitercik. How much data is sufficient to learn high-performing algorithms? Generalization guarantees for data-driven algorithm design. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing (STOC 2021)*, pages 919–932. ACM, 2021a. ↰ p.2

M.-F. Balcan, S. Prasad, and T. Sandholm. Sample complexity of tree search configuration: Cutting planes and beyond. In *Advances in Neural Information Processing Systems (NeurIPS 2021)*, volume 34, pages 4015–4027. Curran Associates, Inc., 2021b. ↰ p.2

M.-F. Balcan, S. Prasad, T. Sandholm, and E. Vitercik. Improved sample complexity bounds for branch-and-cut. In *Proceedings of the 28th International Conference on Principles and Practice of Constraint Programming (CP 2022)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. ↰ p.2

P. Bartlett, V. Maiorov, and R. Meir. Almost linear VC dimension bounds for piecewise polynomial networks. In *Advances in Neural Information Processing Systems (NeurIPS 1998)*, volume 11. MIT Press, 1998. ↰ p.8

P. Bartlett, P. Indyk, and T. Wagner. Generalization bounds for data-driven numerical linear algebra. In *Proceedings of 35th Conference on Learning Theory (COLT 2022)*, volume 178, pages 2013–2040. PMLR, 2022. ↰ p.1, ↰ p.2, ↰ p.3, ↰ p.4, ↰ p.6, ↰ p.8

J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang. JAX: Composable transformations of Python+NumPy programs, 2018. ↰ p.7

K. L. Clarkson and D. P. Woodruff. Numerical linear algebra in the streaming model. In *Proceedings of the 41st annual ACM Symposium on Theory of Computing (STOC 2009)*, pages 205–214. ACM, 2009. ↰ p.1, ↰ p.3

K. L. Clarkson and D. P. Woodruff. Low-rank approximation and regression in input sparsity time. *J. ACM*, 63(6): 1–45, 2017. ↰ p.1, ↰ p.3

L. Csanky. Fast parallel matrix inversion algorithms. *SIAM J. Comput.*, 5(4):618–623, 1976. ↰ p.4

H. P. Decell, Jr. An application of the Cayley–Hamilton theorem to generalized matrix inversion. *SIAM Review*, 7 (4):526–528, 1965. ↰ p.5

A. Gittens and M. W. Mahoney. Revisiting the Nyström method for improved large-scale machine learning. *J. Mach. Learn. Res.*, 17(117):1–65, 2016. ↰ p.5

P. W. Goldberg and M. R. Jerrum. Bounding the Vapnik–Chervonenkis dimension of concept classes parameterized by real numbers. *Mach. Learn.*, 18(2):131–148, 1995. ↰ p.3, ↰ p.4

G. H. Golub and C. F. Van Loan. *Matrix Computation*. The Johns Hopkins University Press, 4th edition, 2013. ↰ p.3

R. Gupta and T. Roughgarden. A PAC approach to application-specific algorithm selection. *SIAM J. Comput.*, pages 123–134, 2017. ↰ p.2

P. Indyk, A. Vakilian, and Y. Yuan. Learning-based low-rank approximations. In *Advances in Neural Information Processing Systems (NeruIPS 2019)*, volume 32. Curran Associates, Inc., 2019. ↰ p.1, ↰ p.2, ↰ p.3, ↰ p.7

P. Indyk, T. Wagner, and D. Woodruff. Few-shot data-driven algorithms for low rank approximation. In *Advances in Neural Information Processing Systems (NeurIPS 2021)*, volume 34, pages 10678–10690. Curran Associates, Inc., 2021. ↰ p.1, ↰ p.2, ↰ p.3, ↰ p.7, ↰ p.8

P. Jain and P. Kar. Non-convex optimization for machine learning. *Foundations and Trends® in Machine Learning*, 10(3-4):142–363, 2017. ↰ p.7

Y. Li, H. Lin, S. Liu, A. Vakilian, and D. Woodruff. Learning the positions in CountSketch. In *International Conference on Learning Representations (ICLR 2023)*, 2023. ↰ p.7

S. Liu, T. Liu, A. Vakilian, Y. Wan, and D. P. Woodruff. Learning the positions in CountSketch. *arXiv:2007.09890*, 2020. ↰ p.1, ↰ p.3, ↰ p.7

P.-G. Martinsson and J. A. Tropp. Randomized numerical linear algebra: Foundations and algorithms. *Acta Numer.*, 29:403–572, 2020. ↰ p.1

E. J. Nyström. Über die praktische auflösung von integralgleichungen mit anwendungen auf randwertaufgaben. *Acta Math.*, 54:185–204, 1930. ↰ p.5

S. Sakaue and T. Oki. Sample complexity of learning heuristic functions for greedy-best-first and A* search. In *Advances in Neural Information Processing Systems (NeurIPS 2022)*, 2022. ↰ p.2

T. Sarlos. Improved approximation algorithms for large matrices via random projections. In *Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2006)*, pages 143–152. IEEE, 2006. ↰ p.1, ↰ p.3

H. E. Warren. Lower bounds for approximation by nonlinear manifolds. *Trans. Amer. Math. Soc.*, 133(1):167–178, 1968. ↰ p.4

D. P. Woodruff. Sketching as a tool for numerical linear algebra. *Foundations and Trends® in Machine Learning*, 10(1–2):1–157, 2014. ↰ p.1, ↰ p.3