Generalization Bounds for Data-Driven Numerical Linear Algebra

Peter Bartlett
UC Berkeley

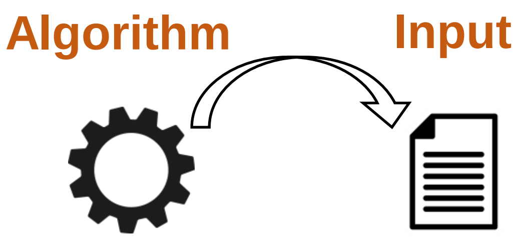
Piotr Indyk
MIT

Tal Wagner
Microsoft Research

COLT 2022

Data-Driven Algorithms

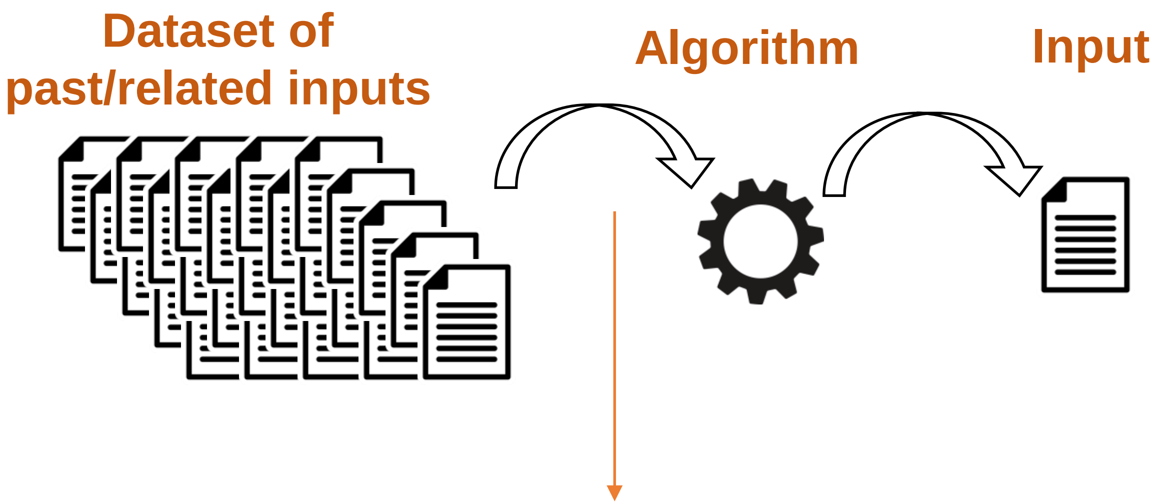
Traditional algorithm design:



Algorithm

Input

Modern reality of algorithm design:



Dataset of past/related inputs

Algorithm

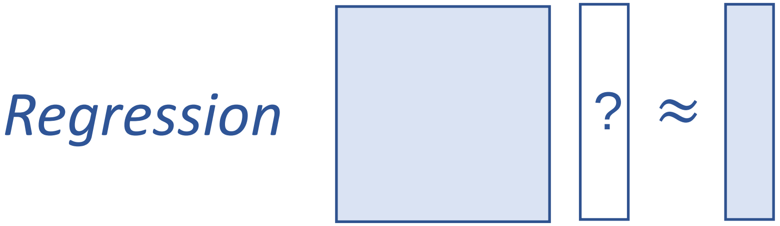
Input

Goal:

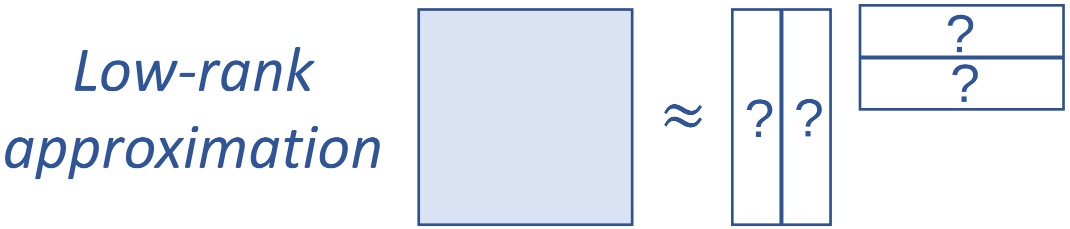
- Use data to improve algorithm
- Automate using ML

Numerical (or Efficient) Linear Algebra

- Problems in computational linear algebra:



Regression

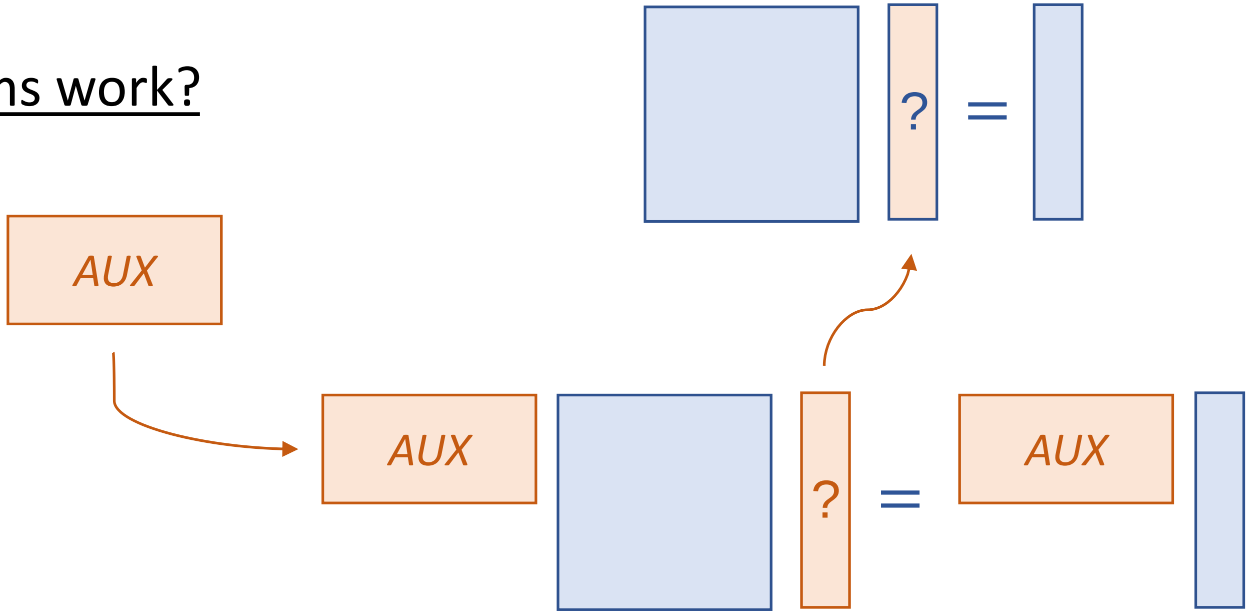


Low-rank approximation

- Exact algorithms: SLOW 🥱
  - $\Omega(n^{\omega})$ for an $n \times n$ matrix
- Approximate algorithms: Near-linear time! 😎
  - $\tilde{O}(n^2)$ for an $n \times n$ matrix, $\tilde{O}(\#nonzero\ entries)$ for a sparse matrix

Data-Driven Numerical Linear Algebra

- How do numerical linear algebra algorithms work?
  - Choose auxiliary matrix
  - Use it to make problem smaller
  - Solve small problem
  - Use solution for large problem

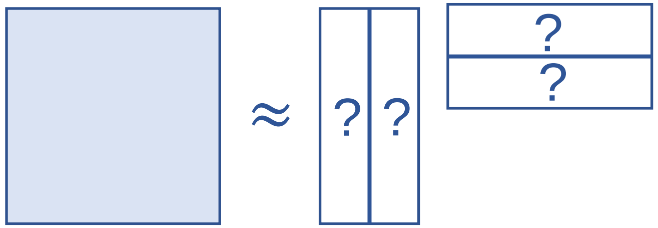


- How do we choose the auxiliary matrix?
  - Traditionally: Either at random 🤪 or by elaborate heuristics
  - Recently: **Learn it from data** (i.e., past inputs) 🤓
    - [Indyk-Vakilian-Yuan’19] [Ailon-Leibovich-Nair’20] [Luz-Galun-Maron-Basri-Yavneh’20] [Liu-Liu-Vakilian-Wan-Woodruff’20] [Indyk-Wagner-Woodruff’21]

Data-Driven Numerical Linear Algebra: In Action

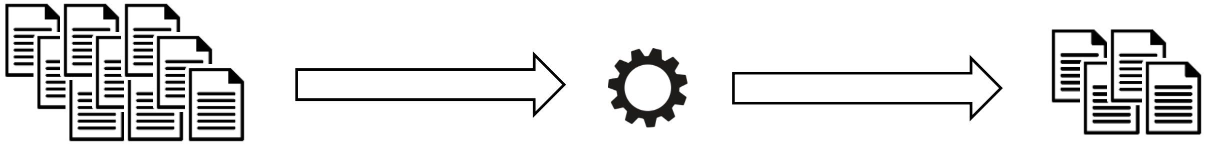
Problem:

Low-rank approximation (LRA)



Setting:

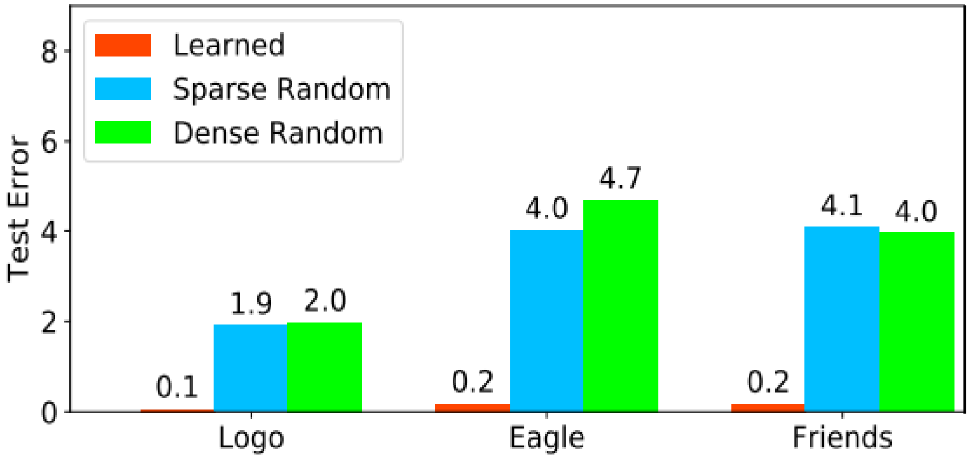
Learning the auxiliary matrix



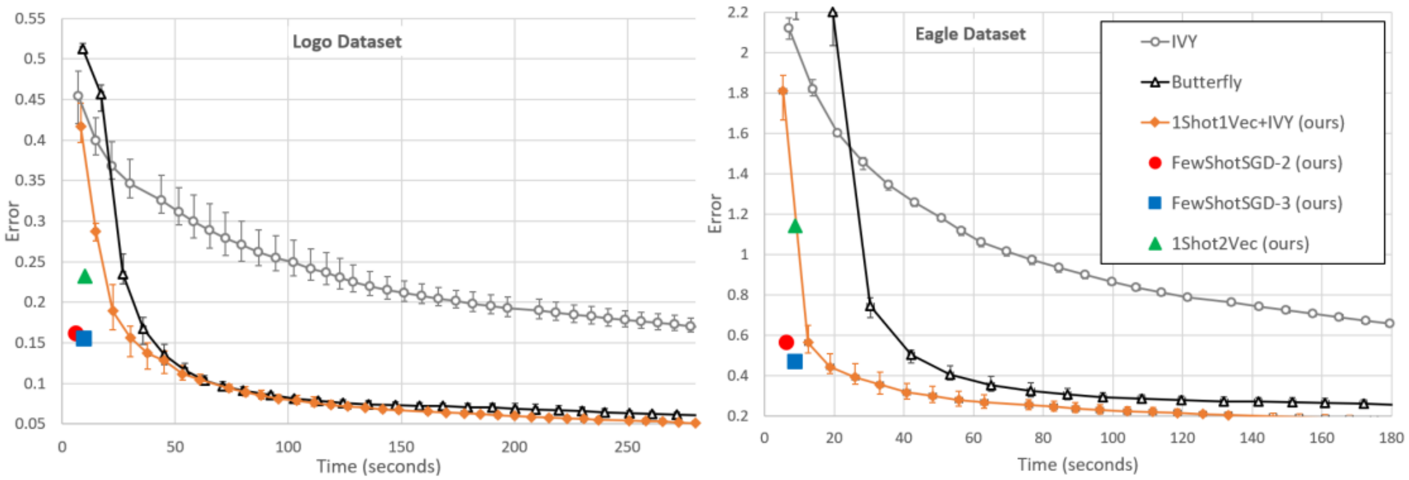
Given a training set of input matrices

Use it to learn an auxiliary matrix ***S***

Evaluate ***S*** by using it for fast (near-linear time) LRA on a test set of input matrices



[Indyk-Vakilian-Yuan NeurIPS’19]



[Indyk-Wagner-Woodruff NeurIPS’21]

Data-Driven Algorithms: Theory?

- Can we **provably** learn good algorithms from past inputs?
- Gupta & Roughgarden (2016):
  - View as statistical learning problem
  - Prove upper bounds on (real-valued analogs of) VC dimension

    ⇒ PAC-learning generalization bounds on number of training samples

- **This work**: Bounds for all existing data-driven numerical linear algebra algorithms

Reference	Algorithm	Problem	Algorithmic "family"
[Indyk-Vakilian-Yuan'19]	**IVY**	LRA	Sketching
[Ailon-Leibovich-Nair'20]	**Butterfly LRA**	LRA	Sketching
[Liu-Liu-Vakilian-Wan-Woodruff'20]	**Multisketch LRA**	LRA	Sketching
[Indyk-Wagner-Woodruff'21]	**Few-shot LRA**	LRA	Sketching
[Luz-Galun-Maron-Basri-Yavneh'20]	**Learned AMG**	Regression	Multigrid

** All bounds are near-proportional to the number of learned parameters*

Prior and Related Work

- Gupta & Roughgarden (ITCS 2016, SICOMP 2017):
  - Initiated framework
  - Upper bound technique for **greedy heuristics** and **local search** algorithms
- Balcan, DeBlasio, Dick, Kingsford, Sandholm, Vitercik (STOC 2021):
  - General upper bound technique
  - Applications for **pattern matching** and **mechanism design** algorithms
  - Does not work for the linear algebra algorithms we consider

Review:

Statistical Learning

(for data-driven algorithms, but also in general)

Data-Driven Algorithms: Setting

A loss minimization problem:

- Inputs: $x \in X$
- Algorithms: $\mathcal{L} = \{L_\rho : \rho \in \mathbb{R}^n\}$, parameterized by $\rho \in \mathbb{R}^n$
- Losses: Identify L_ρ with a map $L_\rho : X \to [0,1]$ that maps inputs to losses
  - $L_\rho(x)$ is the loss of solving for x with parameters ρ

Our case: The low-rank approximation (LRA) problem

- Inputs: X is the set of matrices $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ with $||\boldsymbol{A}||_F = 1$
- Algorithms: $\mathcal{L} = \{L_{\boldsymbol{S}} : \boldsymbol{S} \in \mathbb{R}^{m \times n}\}$, parameterized by auxiliary matrices $\boldsymbol{S} \in \mathbb{R}^{m \times n}$
- Loss: $L_{\boldsymbol{S}}(\boldsymbol{A}) = \left\|\boldsymbol{A} - \widetilde{\boldsymbol{A}}_{\boldsymbol{S}}\right\|_F^2$, where $\widetilde{\boldsymbol{A}}_{\boldsymbol{S}}$ is the LRA of $\boldsymbol{A}$ computed with aux. matrix $\boldsymbol{S}$

Statistical Learning and ERM

Statistical learning: Suppose we have a distribution D over X

- Goal: Estimate the best parameters for D

$$\rho^* = \mathrm{argmin}_{\rho \in \mathbb{R}^n} \mathbb{E}_{x \in D}\left[L_\rho(x)\right]$$

- Method: Draw s samples $x_1, \ldots, x_s \sim D$ and use Empirical Risk Minimization (ERM)

$$\hat{\rho} = \mathrm{argmin}_{\rho \in \mathbb{R}^n} \frac{1}{s} \sum_{i=1}^{s} L_\rho(x_i)$$

- We say $\mathcal{L} = \{L_\rho : \rho \in \mathbb{R}^n\}$ is (ϵ, δ)-*learnable* with s samples (by ERM) if

$$\Pr_{x_1,\ldots,x_s \sim D}\left[\mathbb{E}_{x \in D}\left[L_{\hat{\rho}}(x)\right] \le \mathbb{E}_{x \in D}\left[L_{\rho^*}(x)\right] + \epsilon\right] \ge 1 - \delta$$

- ***Question***: *What is the smallest number of samples s that suffices?*

VC-Dimension and Fat Shattering Dimension

Definition: Let $\mathcal{L}$ be a family of functions $X \to \{0,1\}$.

- A set $x_1, \dots, x_s \in X$ is **shattered** by $\mathcal{L}$ if for every $I \subset \{1, \dots, s\}$, there is $L \in \mathcal{L}$ s.t.:

$$L(x_i) = 1 \Leftrightarrow i \in I.$$

- The **VC-dimension** $\mathrm{VCdim}(\mathcal{L})$ of $\mathcal{L}$ is the size of the largest shattered set.

Definition: Let $\mathcal{L}$ be a family of functions $X \to [0,1]$. Let $\gamma \geq 0$.

- A set $x_1, \dots, x_s \in X$ is γ**-fat shattered** by $\mathcal{L}$ if there are thresholds $r_1, \dots, r_s \in \mathbb{R}$, such that for every $I \subset \{1, \dots, s\}$, there is $L \in \mathcal{L}$ s.t.:

$$i \in I \Rightarrow L(x_i) > r_i + \gamma \quad \text{and} \quad i \notin I \Rightarrow L(x_i) < r_i - \gamma$$

- The γ**-fat shattering dimension** $\mathrm{fat}_\gamma(\mathcal{L})$ of $\mathcal{L}$ is the size of the largest γ-fat shattered set.

Classical learning theory: The sample complexity of (ϵ, δ)-learning $\mathcal{L}$ (by ERM) is proportional to the γ-fat shattering dimension with $\gamma = \Theta(\epsilon)$.

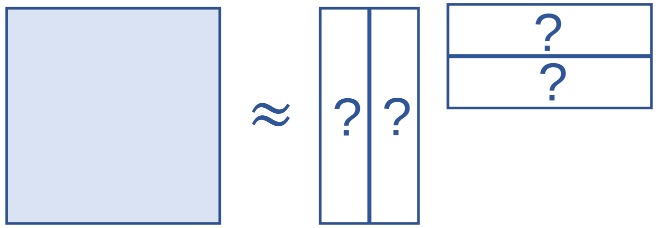
Review:

Fast Low-Rank Approximation

(with data-driven algorithms, but also in general)

Low-Rank Approximation (LRA)

- **Problem:**
  - Input: $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ with $\|\boldsymbol{A}\|_F = 1$, target rank $k \ll n$
  - Goal: $\widetilde{\boldsymbol{A}}$ of rank k that approximately minimizes $\left\|\boldsymbol{A} - \widetilde{\boldsymbol{A}}\right\|_F^2$

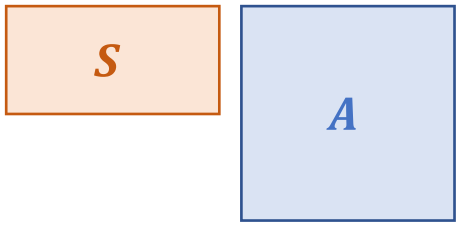


- **Exact solution: SVD**
  - Returns: $[\boldsymbol{A}]_k$ such that $\|\boldsymbol{A} - [\boldsymbol{A}]_k\|_F^2 = \min_{\widetilde{\boldsymbol{A}} \text{ of rank } k} \left\|\boldsymbol{A} - \widetilde{\boldsymbol{A}}\right\|_F^2$
  - Runtime: $O(n^{\omega})$ 🥱

Efficient Low-Rank Approximation

- The **SCW** algorithm [Sarlos'06, Clarkson-Woodruff'09,13]:
  - Pick an auxiliary matrix $\boldsymbol{S} \in \mathbb{R}^{m \times n}$, where $k \leq m \ll n$
  - Project $\boldsymbol{A}$ onto $rows(\boldsymbol{SA})$
  - Return: Best rank-k approximation of projected $\boldsymbol{A}$

$$\widetilde{\boldsymbol{A}}_{\boldsymbol{S}} = \left[\boldsymbol{A}(\boldsymbol{SA})^{\dagger}(\boldsymbol{SA})\right]_k$$

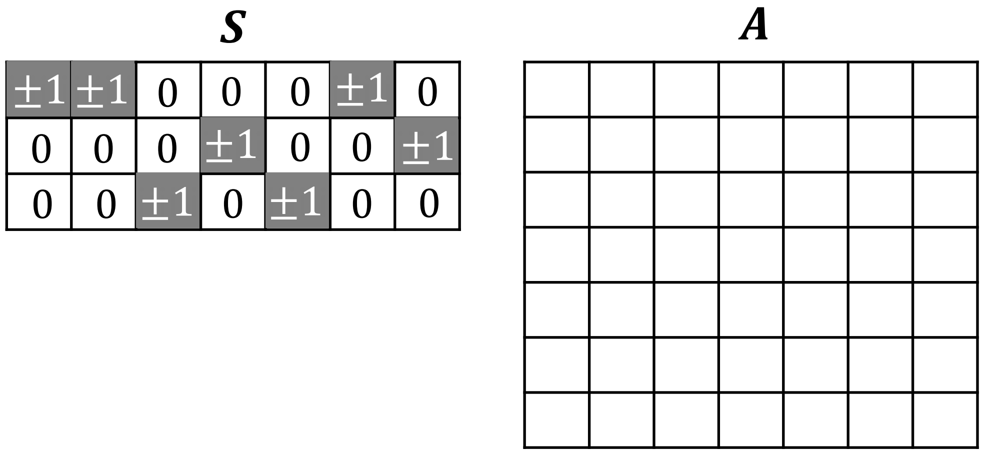


Reminder: $\boldsymbol{M}^{\dagger}\boldsymbol{M}$ is the orthogonal projection matrix on the row space $\boldsymbol{M}$

- Lemma: $\widetilde{\boldsymbol{A}}_{\boldsymbol{S}}$ can be computed in time $mult(\boldsymbol{S}, \boldsymbol{A}) + O(m^2 n)$ and space $O(mn)$.
  - By the formula: $\widetilde{\boldsymbol{A}}_{\boldsymbol{S}} = [\boldsymbol{AV}]_k \boldsymbol{V}^T$ where $\boldsymbol{SA} = \boldsymbol{U\Sigma V}^T$

- Theorem: If $\boldsymbol{S}$ has $m = \tilde{O}(k/\epsilon)^2$ rows s.t.:
  - Each column one uniformly random non-zero
  - Each non-zero is uniform in $\{1, -1\}$

Then, whp, $\left\|\boldsymbol{A} - \widetilde{\boldsymbol{A}}_{\boldsymbol{S}}\right\|_F^2 \leq (1 + \epsilon) \cdot \|\boldsymbol{A} - [\boldsymbol{A}]_k\|_F^2$.



$$\boldsymbol{S} = \begin{array}{|c|c|c|c|c|c|c|} \hline \pm1 & \pm1 & 0 & 0 & 0 & \pm1 & 0 \\ \hline 0 & 0 & 0 & \pm1 & 0 & 0 & \pm1 \\ \hline 0 & 0 & \pm1 & 0 & \pm1 & 0 & 0 \\ \hline \end{array}$$

Data-Driven Low-Rank Approximation

- The **SCW** algorithm: $\mathrm{SCW}_k(\boldsymbol{S},\boldsymbol{A}) = \left[\boldsymbol{A}(\boldsymbol{SA})^{\dagger}(\boldsymbol{SA})\right]_k$
  - Loss: $L_{\boldsymbol{S}}(\boldsymbol{A}) = \|\boldsymbol{A} - \mathrm{SCW}_k(\boldsymbol{S},\boldsymbol{A})\|_F^2$

- **Oblivious** auxiliary matrix $\boldsymbol{S} \in \mathbb{R}^{m\times n}$:
  - Each column one uniformly random non-zero
  - Each non-zero is uniform in $\{1,-1\}$

$$\boldsymbol{S} = \begin{bmatrix} 1 & -1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & -1 & 0 & 0 \end{bmatrix}$$

- **Data-driven** auxiliary matrix $\boldsymbol{S} \in \mathbb{R}^{m\times n}$ [Indyk-Vakilian-Yuan’19]:
  - Each column one uniformly random non-zero
  - Its value is a trainable parameter (learned via SGD)

$$\boldsymbol{S} = \begin{bmatrix} \rho_1 & \rho_2 & 0 & 0 & 0 & \rho_6 & 0 \\ 0 & 0 & 0 & \rho_4 & 0 & 0 & \rho_7 \\ 0 & 0 & \rho_3 & 0 & \rho_5 & 0 & 0 \end{bmatrix}$$

- **How many samples do we need to ERM-learn $\{L_{\boldsymbol{S}}(\boldsymbol{A})\}$?**

Our Results

Theorem – Fat shattering dimension of SCW:

- Upper bound: The ϵ-fat shattering dimension of learned SCW is
$$O\big(n \cdot (m + k\log(n/k) + \log(1/\epsilon))\big),$$
with $\boldsymbol{A} \in \mathbb{R}^{n\times n}$, $\boldsymbol{S} \in \mathbb{R}^{m\times n}$, and low rank k.
- Lower bound: The ϵ-fat shattering dimension of learned SCW is $\Omega(n)$, if $\epsilon < 1/(2\sqrt{k})$.
- Techniques apply to all other existing data-driven linear algebra algorithms.

Corollary – Sample complexity of learning SCW:

- Learning SCW with ERM requires at most $\tilde{O}(\epsilon^{-2} n \cdot m)$ samples.
- Learning SCW with ERM requires at least $\Omega(\epsilon^{-2}\, n/k)$ samples.
- Learning SCW with any method requires at least $\Omega(n + \epsilon^{-1})$ samples.

Remarks

- What if training set has nothing to do with test set?
  - **Safeguarding** [Indyk-Vakilian-Yuan'19]:
    - Vertically augment the learned $\boldsymbol{S}$ a random $\boldsymbol{S}'$
    - Number of rows is only doubled
    - Guarantees: $\boldsymbol{S}$ is at least as good as $\boldsymbol{S}'$ on any input matrix $\boldsymbol{A}$

- What about the running time of computing the best $\boldsymbol{S}$ for the sample?
  - In practice: Use stochastic gradient descent (SGD) on training set
  - Not known to provably converge to empirical risk minimizer

Proof Overview

The Goldberg-Jerrum'95 Framework

Definition: A **GJ-algorithm** is a deterministic algorithm on real-valued inputs, with two types of operations:

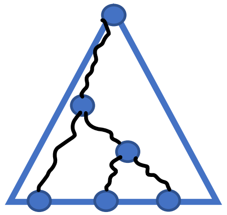
- Arithmetic: $v'' = v \odot v'$ where $\odot \in \{+, -, \times, \div\}$
- Conditional: "if $v \geq 0$ then … else …"

Where v, v' are either inputs or values previously computed by the algorithm.

Theorem [Goldberg-Jerrum'95]:

- Suppose there is a GJ-algorithm that takes $x \in X$, $\rho \in \mathbb{R}^n$ and $r \in \mathbb{R}$, and returns TRUE iff $L_\rho(x) \geq r$. Suppose it has running time T.
- Then, $\forall \gamma$, the γ-fat shattering dimension of $\mathcal{L} = \{L_\rho : \rho \in \mathbb{R}^n\}$ is $O(nT)$.

Proof sketch: The GJ algorithm partitions $\mathbb{R}^n$ into constant sign regions with polynomial boundaries. Classical theorems on polynomials [Milnor'64, Warren'68] bound the number of sign regions.



Goldberg-Jerrum: First Attempt

- Goal: GJ-algorithm for the SCW loss, $L_{\boldsymbol{S}}(\boldsymbol{A}) = \left\| \boldsymbol{A} - \left[\boldsymbol{A}(\boldsymbol{SA})^{\dagger}(\boldsymbol{SA}) \right]_k \right\|_F^2$.
- Need two steps:
  1. Projection (compute $\boldsymbol{M}^{\dagger}\boldsymbol{M}$ for a matrix $\boldsymbol{M}$)
  2. Best rank-k approximation (computing $[\boldsymbol{M}]_k$ for a matrix $\boldsymbol{M}$)

- Problem: How to compute $[\boldsymbol{M}]_k$ with a GJ-algorithm (only arithmetic operations)?
- Solution: Approximate by the Power Method
  - $[\boldsymbol{M}]_k \approx \boldsymbol{Z}\boldsymbol{Z}^{\dagger}\boldsymbol{M}$ with $\boldsymbol{Z} = (\boldsymbol{M}\boldsymbol{M}^T)^q \boldsymbol{M}\boldsymbol{\Pi}$ and gaussian $\boldsymbol{\Pi} \in \mathbb{R}^{n \times k}$

[Rokhlin-Szlam-Tygert'10, Halko-Martinsson-Tropp'11, Boutsidis-Drineas-MagdonIsmail'14, Woodruff'14, Witten-Candes'15, Musco-Musco'15]

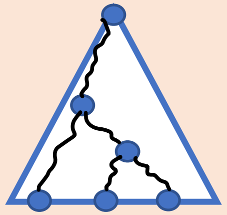
- However:
  - Power Method is *slow* 🥱 q iterations take $qkn^{O(1)}$ time (here $q = O(\epsilon^{-1}\log(n/\epsilon))$)
  - Power Method is *randomized* 🤪 derandomizing $\boldsymbol{\Pi}$ takes $(n/k)^k$ time

Refined Goldberg-Jerrum: Definitions

Definition: A **GJ-algorithm** is a deterministic algorithm on real-valued inputs, with two types of operations:

- Arithmetic: $v'' = v \odot v'$ where $\odot \in \{+,-,\times,\div\}$
- Conditional: "if $v \geq 0$ then … else …"

Where v, v' are either inputs or values previously computed by the algorithm.



Observe: Every value computed by a GJ-algorithm is a rational function of the inputs.

Definition:

- The **degree** of a GJ-algorithm is the maximum degree of any rational function it computes.
- The **predicate complexity** of a GJ-algorithm is the number of ***distinct*** rational functions in its conditional statements.

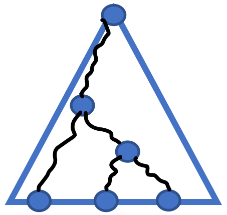
Refined Goldberg-Jerrum: Theorem

Theorem: Suppose we have,

- A GJ-algorithm that takes $x \in X$, $\rho \in \mathbb{R}^n$ and $r \in \mathbb{R}$, and returns TRUE iff $L_\rho(x) \geq r$.
- Suppose it has **degree** Δ and **predicate complexity** P.
- Then, $\forall \epsilon$, the ϵ-fat shattering dimension of $\mathcal{L} = \{L_\rho : \rho \in \mathbb{R}^n\}$ is $O(n \log(\Delta P))$.

Observe: Runtime T implies $\Delta, P \leq 2^T$.

- Thus, refines the previous theorem, *“runtime* $T \Rightarrow$ *fat-dim* $O(nT)$*”.*



Why does it help?

- q Power Method iterations: time $qkn^{O(1)}$, **but** degree $O(q)$
  - $nT = qkn^{O(1)}$ **but** $n \log \Delta = O(n \log q)$
- Minimum of t numbers: time $O(t)$, **but** predicate complexity $\binom{t}{2}$
  - $nT = O(nt)$ **but** $n \log P = O(n \log t)$
  - Derandomizing the power method: $t = \binom{n}{k}$, thus $nT = O\left(n(n/k)^k\right)$ **but** $n \log P = O(nk \log(n/k))$

Refined GJ-Algorithm for SCW

- New goal:
  - GJ-algorithm for the SCW loss, $L_{\boldsymbol{S}}(\boldsymbol{A}) = \left\| \boldsymbol{A} - \left[\boldsymbol{A}(\boldsymbol{SA})^{\dagger}(\boldsymbol{SA}) \right]_k \right\|_F^2$
  - With efficient **degree** and **predicate complexity**.

- Need two steps:
  1. Projection (compute $\boldsymbol{M}^{\dagger}\boldsymbol{M}$ for a matrix $\boldsymbol{M}$)
  2. Best rank-k approximation (computing $[\boldsymbol{M}]_k$ for a matrix $\boldsymbol{M}$)

Step 1: Computing Projection Matrices

Lemma 1: Given $\boldsymbol{M} \in \mathbb{R}^{m \times n}$, there is a GJ-algorithm for computing $\boldsymbol{M}^{\dagger}\boldsymbol{M}$ with degree $O(m)$ and predicate complexity 2^m.

Proof:

- Fact 1: If the rows of $\boldsymbol{N}$ form a basis for the rows of $\boldsymbol{M}$, then $\boldsymbol{M}^{\dagger}\boldsymbol{M} = \boldsymbol{N}^T(\boldsymbol{N}\boldsymbol{N}^T)^{-1}\boldsymbol{N}$.
- Fact 2 (e.g., [Csanky'76]): There are algorithms that use only arithmetic operations for

  (i) checking if a matrix has full rank,

  (ii) inverting an invertible matrix.

  Their degree (as GJ-algorithms) for $m \times m$ matrices is $O(m)$.
- **GJ-Algorithm**: Try all 2^m subsets of rows of $\boldsymbol{M}$ (predicate complexity 2^m) to find a basis $\boldsymbol{N}$ (use Fact 2**(i)** to check it is a basis). Invert $\boldsymbol{N}\boldsymbol{N}^T$ (with Fact 2**(ii)**). Return $\boldsymbol{N}^T(\boldsymbol{N}\boldsymbol{N}^T)^{-1}\boldsymbol{N}$.

■

Step 2: Derandomized Power Method

Lemma 2: Given $\boldsymbol{M} \in \mathbb{R}^{n\times n}$, there is a GJ-algorithm for ϵ-approximating $[\boldsymbol{M}]_k$, with degree $O(k\epsilon^{-1}\log(n/\epsilon\,))$ and predicate complexity $2^k\binom{n}{k}^2$.

Remark: Approximation is why we needed the gap γ in the definition of fat shattering.

Proof (sketch):

- Fact 3: Starting with a gaussian $\boldsymbol{\Pi} \in \mathbb{R}^{n\times k}$, $q = O(\epsilon^{-1}\log(n/\epsilon\,))$ Power Methods iterations suffice to ϵ-approximate $[\boldsymbol{M}]_k$ [Musco-Musco'15], by the formula:

$$[\boldsymbol{M}]_k \approx_\epsilon \boldsymbol{Z}\boldsymbol{Z}^\dagger\boldsymbol{M} \text{ where } \boldsymbol{Z} = (\boldsymbol{M}\boldsymbol{M}^T)^q\boldsymbol{M}\boldsymbol{\Pi}$$

- Fact 4: $\boldsymbol{\Pi}$ can be derandomized with k-subsets of the standard basis in $\mathbb{R}^n$.
- **GJ algorithm:** Try all $\binom{n}{k}$ subset of $\mathbb{R}^n$ as the initial matrix $\boldsymbol{\Pi}$. For each one, compute $\boldsymbol{Z}\boldsymbol{Z}^\dagger\boldsymbol{M}$ (using the previous lemma for $\boldsymbol{Z}\boldsymbol{Z}^\dagger$, which blows up the degree by k and the predicate complexity by 2^k). pick the one that minimizes the LRA error $\left\|\boldsymbol{M} - \boldsymbol{Z}\boldsymbol{Z}^\dagger\boldsymbol{M}\right\|_F^2$.