GPU parallelizable sketch-and-precondition

Tyler Chen

JPMorganChase

Disclaimer

Paper

GPU-Parallelizable Randomized Sketch-and-Precondition for Linear Regression using Sparse Sign Sketches

Tyler Chen, Pradeep Niroula, Archan Ray, Pragna Subrahmanya, Marco Pistoia, Niraj Kumar

https://arxiv.org/abs/2506.03070

Linear Regression

We are interested in solving the least squares problem

$$\min_{\mathbf{x}\in\mathbb{R}^n} \|\mathbf{b} - \mathbf{A}\mathbf{x}\|, \quad \mathbf{A} \in \mathbb{R}^{m\times n}, \quad \mathbf{b} \in \mathbb{R}^m, \quad m \gg n \gg 1.$$

Classical factorization methods (e.g. Householder QR):

- require $O(mn^2)$ arithmetic operations
- do not take advantage of sparsity in $\mathbf{A}$

Classical iterative methods (e.g. LSQR):

- require $O(\mathrm{cond}(\mathbf{A})\,\mathrm{nnz}(\mathbf{A})\log(1/\varepsilon))$ arithmetic operations
- intractable if $\mathbf{A}$ is ill-conditioned.

Sketch-and-precondition

We will construct a preconditioner $\mathbf{M} \in \mathbb{R}^{n\times n}$ and solve

$$\min_{\mathbf{x}\in\mathbb{R}^n} \|\mathbf{b} - (\mathbf{AM})\mathbf{y}\|, \qquad \mathbf{x} = \mathbf{My}.$$

If **AM** is well-conditioned, then the convergence of iterative methods is fast!

In general, finding a good preconditioned **M** can be hard. However, using RandNLA, we can efficiently (when $m \gg n$) construct an excellent preconditioner (e.g. for which $\mathrm{cond}(\mathbf{AM}) \leq 10$).[1]

[1] Rokhlin and Tygert 2008; Avron, Maymounkov, and Toledo 2010.

Comment on measuring error

Lots of times, papers give the gurantees like

$$\|\mathbf{b} - \mathbf{A}\widehat{\mathbf{x}}\|^2 \le (1+\varepsilon)\|\mathbf{b} - \mathbf{A}\mathbf{x}_\star\|^2.$$

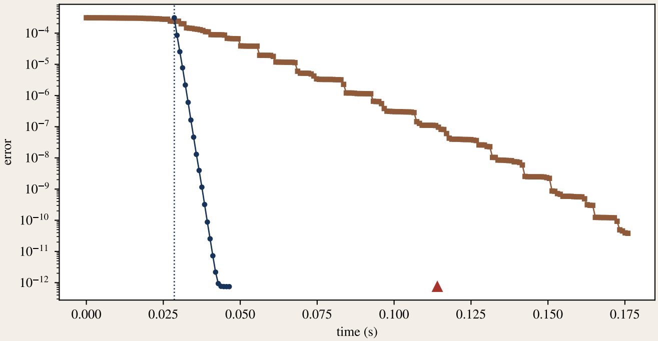
This is equivalent to a characterization in terms of the $\mathbf{A}^\top\mathbf{A}$-norm error:

$$\begin{aligned}\|\mathbf{b} - \mathbf{A}\widehat{\mathbf{x}}\|^2 &= \|\mathbf{b} - \mathbf{A}(\mathbf{x}_\star + \widehat{\mathbf{x}} - \mathbf{x}_\star)\|^2 \\ &= \| \underbrace{\mathbf{b} - \mathbf{A}\mathbf{x}_\star}_{\in \operatorname{span}(\mathbf{A})^\perp} - \underbrace{\mathbf{A}(\mathbf{x}_\star - \widehat{\mathbf{x}})}_{\in \operatorname{span}(\mathbf{A})} \|^2 \\ &= \|\mathbf{b} - \mathbf{A}\mathbf{x}_\star\|^2 + \|\mathbf{A}(\mathbf{x}_\star - \widehat{\mathbf{x}})\|^2.\end{aligned}$$

Rearranging, we find that

$$\|\mathbf{A}(\mathbf{x}_\star - \widehat{\mathbf{x}})\|^2 = \|\mathbf{b} - \mathbf{A}\widehat{\mathbf{x}}\| - \|\mathbf{b} - \mathbf{A}\mathbf{x}_\star\|^2 \le \varepsilon\, \|\mathbf{b} - \mathbf{A}\mathbf{x}_\star\|^2.$$

Example



Legend: Error as a function of time for direct solver (▲), iterative method without a preconditioner (—■—), and sketch-and-precondition (—●—).

Subspace embedding

Definition. Let $V \subset \mathbb{R}^m$ be a subspace. We say $\mathbf{S} \in \mathbb{R}^{d \times m}$ is an subspace embedding for V with distortion η if

$$\forall \mathbf{z} \in V : \quad (1-\eta)\|\mathbf{z}\| \leq \|\mathbf{S}\mathbf{z}\| \leq (1+\eta)\|\mathbf{z}\|.$$

If we get a subspace embedding for $\mathrm{range}(\mathbf{A}, \mathbf{b})$ then we can efficiently construct:

- a good preconditioner $\mathbf{M}$
- a good initial guess $\mathbf{x}_0$

Together, these will let us apply LSQR for a small number of iterations.

For this to be efficient we need:

- $d \ll m$ (so that $\mathbf{SA}$ is easier to process than $\mathbf{A}$)
- $\mathbf{S}$ is efficient to generate and apply

More precise statement

If $\mathbf{S}$ is a subspace embedding for $\mathbf{A}$ with distortion η. Then,

$$\mathrm{cond}(\mathbf{AM}) \le \frac{1+\eta}{1-\eta}$$

where $\mathbf{M}$ is such that $\mathbf{SAM}$ has orthonormal columns. Moreover, if $\mathbf{S}$ is a subspace embedding for $(\mathbf{A}, \mathbf{b})$ with distortion η, then (coarse bound)

$$\|\mathbf{A}(\mathbf{x}_\star - \mathbf{x}_0)\| \le \left(\frac{2\eta}{1-\eta}\right)^{1/2} \|\mathbf{b} - \mathbf{A}\mathbf{x}_\star\|,$$

where $\mathbf{x}_0$ is the solution to the sketched problem $\min_{\mathbf{x}} \|\mathbf{Sb} - \mathbf{SAx}\|$.

So, we can set η as something constant (e.g. $\eta = 1/2$)!

How does embedding dimension relate to distorition?

Let $\mathbf{V}$ be an onb for V. Subspace embedding:

$$\forall \mathbf{c} \in \mathbb{R}^n : \quad (1-\eta)\|\mathbf{Vc}\| \le \|\mathbf{SVc}\| \le (1+\eta)\|\mathbf{Vc}\|$$

So we care about

$$\max_{\mathbf{c}\in\mathbb{R}^n} \frac{\|\mathbf{SVc}\|}{\|\mathbf{Vc}\|} = \max_{\mathbf{c}\in\mathbb{R}^n} \frac{\|\mathbf{SVc}\|}{\|\mathbf{c}\|} = \sigma_{\max}(\mathbf{SV}), \quad \min_{\mathbf{c}\in\mathbb{R}^n} \frac{\|\mathbf{SVc}\|}{\|\mathbf{Vc}\|} = \min_{\mathbf{c}\in\mathbb{R}^n} \frac{\|\mathbf{SVc}\|}{\|\mathbf{c}\|} = \sigma_{\min}(\mathbf{SV}).$$

If $\mathbf{S}$ is Gaussian, then so is $\mathbf{SV}$ (by orthogonal invariance). Up to scaling,

$$\sigma_{\max}(\mathbf{SV}) \approx 1 + \sqrt{d/n}, \qquad \sigma_{\min}(\mathbf{SV}) \approx 1 - \sqrt{d/n}.$$

Many “mixing” sketches behave like Gaussians, so $\eta \approx \sqrt{d/n}$ is good reference.

Sketch and precondition

This gives the sketch-and-precondition algorithm (Rokhlin and Tygert 2008).

Algorithm:

1. Generate $\mathbf{S}$ (choose hyperparameters)
2. Compute $\mathbf{SA}$, $\mathbf{Sb}$
3. Factor $\mathbf{QR} = \mathbf{SA}$ (or SVD)
4. Get initial guess $\mathbf{x}_0 = \mathbf{R}^{-1}\mathbf{Q}^\top\mathbf{Sb}$
5. Preconditioner $\mathbf{M} = \mathbf{R}^{-1}$
6. Run preconditioned iterative method

Sparse sign sketch

Definition: We say $\mathbf{S} \in \mathbb{R}^{d\times m}$ is a sparse sign sketching matrix with sparsity parameter ζ if

$$\mathbf{S} = \sqrt{\frac{m}{\zeta}} \begin{bmatrix} \mathbf{s}_1 & \mathbf{s}_2 & \cdots & \mathbf{s}_m \end{bmatrix},$$

where each column $\mathbf{s}_i$ is independent and consists of exactly ζ random signs situated in uniformly random coordinates.[2]

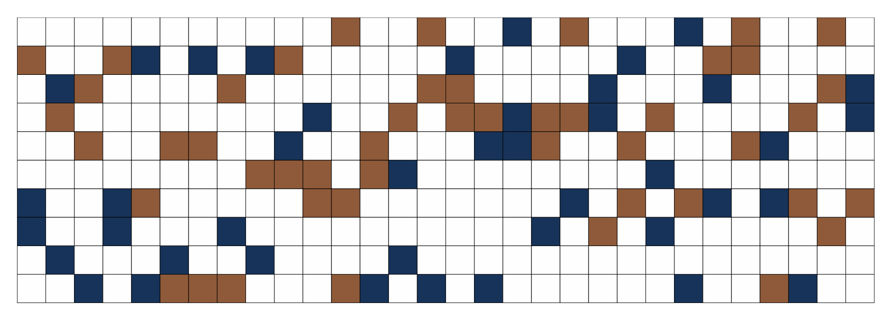
Theorem (Cohen 2015): Subspace embedding with distortion η if $d = O(n\log(n)/\eta^2)$ and $\zeta = O(\log(n)/\eta)$.

Other sparse sketches: More at the end..

[2] This is often called CountSketch when $\zeta = 1$.

Sparse sign sketch

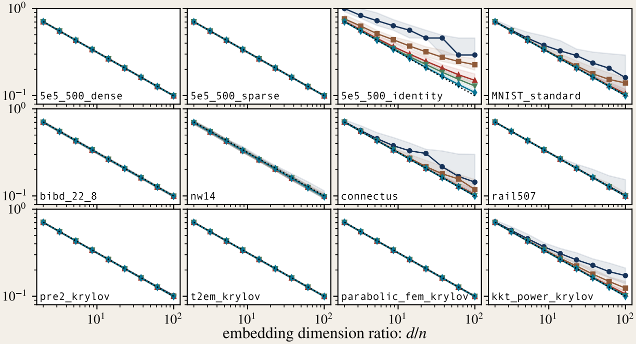
Illusration of sparse sign matrix with $d = 10$, $m = 30$, and $\zeta = 3$:



Legend: gold/light entries $= +1$, blue/dark entries $= -1$

Sparse sign sketch quality

Sparse sign sketches (even with small ζ) behave similarly to Gaussian sketches.



Legend: $\zeta = 2$ (●), $\zeta = 4$ (■), $\zeta = 8$ (▲), $\zeta = 12$ (▼), $\zeta = 24$ (◆)

Generating sparse sign sketches

It is straightforward to sample the values of the ζm nonzero entries of **S**, which are independent random (scaled) signs.

The more involved task is determining, in each column, which of the ζ rows will be nonzero. This task is equivalent to sampling a $\zeta \times m$ matrix

$$\mathbf{C} = \begin{bmatrix} c_{1,1} & c_{1,2} & \cdots & c_{1,m} \\ \vdots & \vdots & & \vdots \\ c_{\zeta,1} & c_{\zeta,2} & \cdots & c_{\zeta,m} \end{bmatrix},$$

where each column of **C** independently contains exactly ζ numbers drawn from $[d] = \{1, \ldots, d\}$ without replacement.

A CSC sparse representation can then easily be constructed.

High-level implementation

We want a high-level implementation for generating **C**.

- If we can offload costly operations to low-level primitives, then we get an effecient platform agnostic implementation.
- Efficiently generating sparse sign sketches in python/MATLAB/etc. is non-trivial.

We are unaware of any existing efficient high-level implementations!

Since the columns of **C** are iid, (mathematically) we can consider a single column. However, we should keep in mind that we will want to be able to easily parallelize over columns.

Shuffling algorithms

Naive: Permute $\{1, 2, \ldots, n\}$ and take first ζ entries.

Fisher–Yates:

1. sample an integer x uniformly from $[d - j]$
2. let δ be the number of numbers in C less than or equal to x
3. append $x + \delta$ to C

Naieve implemention requires $O(\zeta^2)$ work per column to check inclusion.

Fisher–Yates (inplace): (used by Murray et al. 2023 for RandLAPACK)

1. initialize list $[1, \ldots, m]$
2. sample an integer x uniformly from $[d - j]$
3. swap indices from j and x

This now requires $O(\zeta)$ work per column (but $O(m)$ working space).

Our approach

Rejection sampling: (used by Epperly 2024)

1. Generate x uniformly from [d]
2. If x is not contained in C append it, else resample

Cost depends on number of rejections, but typically $\zeta \ll$ d.

Our implementation: we use a vectorized version of rejection sampling:

1. sort along axis
2. compare neighbors
3. resample bad indices

This can be efficiently implemented in high-level langaguges.

Numerical Experiments

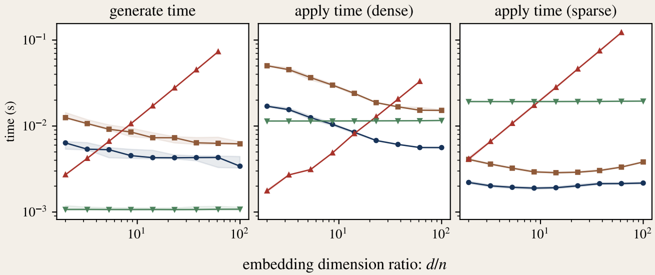
Choice of sketching distribution

There are lots of ways to get a subspace embedding, including oblivious methods. Common choices include:

- Gaussian
- Fast Trigonometric
- Sparse

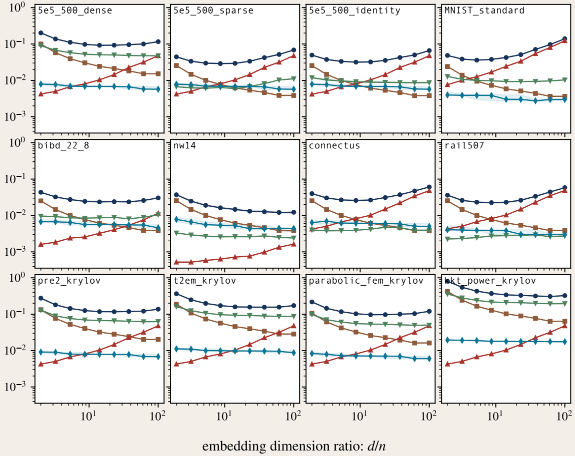
These “mixing sketches” all behave similar to Gaussian with respect to the embedding dimension.

Comparison of generate/apply time



Legend: Generate and apply time (to dense and sparse **A**) for Gaussian ( ), subsampled trig ( ), and sparse sign sketch with $\zeta = 8$ ( ) and $\zeta = 24$ ( ).

Total Runtime



Legend: Total runtime of sketch-and-precondition ($\zeta = 12$) as a function of embedding dimension ( ) and individual components: iteration time ( ), preconditioner build time ( ), sketch apply time ( ), and sketch generate time ( ).

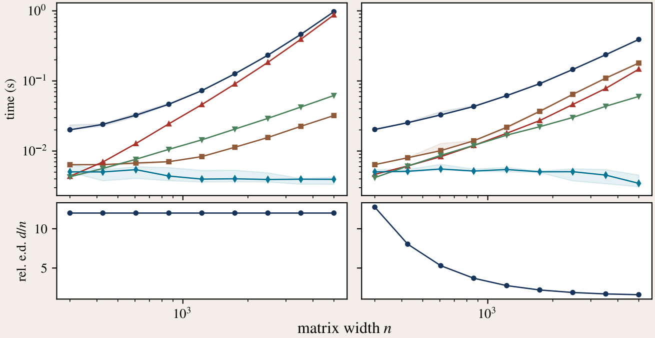
Embedding dimension

How should we select the embedding dimension d (which controls the subspace embedding distorition η)?

- As $\eta \to 0$ the convergence of LSQR is faster, but factoring the sketch is more expensive.

Gaussian sketch gives $\eta \asymp \sqrt{n/d}$, which we can use as a proxy.

Embedding dimension



Legend: Total runtime of sketch-and-precondition ($\zeta = 12$) as a function of embedding dimension ( ) and individual components: iteration time ( ), preconditioner build time ( ), sketch apply time ( ), and sketch generate time ( ).

Multi-GPU computations

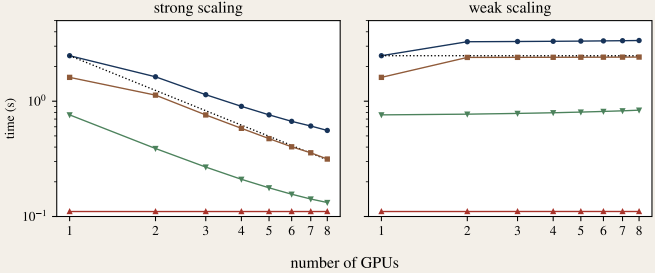
We study the behavior of sketch-and-precondition with sparse sketches when we have multiple GPU devices (8 A100 GPUs on a single node).

Parallelize across long axis (dimension m vectors): Specifically, partition $\{1, \ldots, m\}$ to $I_1, \ldots, I_p$. Each GPU holds some rows $\mathbf{A}[I_\ell, \cdot\,]$ of the data-matrix.

$$\mathbf{Ax} = \begin{bmatrix} \mathbf{A}[I_1, \cdot\,] \\ \vdots \\ \mathbf{A}[I_p, \cdot\,] \end{bmatrix} \mathbf{x} = \begin{bmatrix} \mathbf{A}[I_1, \cdot\,]\mathbf{x} \\ \vdots \\ \mathbf{A}[I_p, \cdot\,]\mathbf{x} \end{bmatrix}$$

$$\mathbf{A}^\mathsf{T}\mathbf{y} = \begin{bmatrix} \mathbf{A}[I_1, \cdot\,] \\ \vdots \\ \mathbf{A}[I_p, \cdot\,] \end{bmatrix}^\mathsf{T} \begin{bmatrix} \mathbf{y}[I_0] \\ \vdots \\ \mathbf{y}[I_p] \end{bmatrix} = \mathbf{A}[I_1, \cdot\,]^\mathsf{T}\mathbf{y}[I_1, \cdot\,] + \cdots + \mathbf{A}[I_p, \cdot\,]^\mathsf{T}\mathbf{y}[I_p].$$

Scaling Experiments



Legend: Total runtime of sketch-and-precondition ($\zeta = 12$) as a function of embedding dimension ( ) and individual components: iteration time ( ), preconditioner build time ( ), sketch apply time ( ), and sketch generate time ( ).

Conclusions

- Sparse sketches seem to outperform other sketching types in most computational settings
- Sparse sign sketches can be efficiently generated in a high-level language (but it requires some work)
- Sparse sign sketches and sketch-and-precondition have very good performance on one and multiple GPU devices (for tall problems)
- Practical implementations of sketch-and-precondition should aim to adaptively determine the embedding dimension

Other sparse sketches

Definition: We say $\mathbf{S} \in \mathbb{R}^{d \times m}$ is a sparse stack sketching matrix with sparsity parameter ζ if

$$\mathbf{S} = \sqrt{\frac{m}{\zeta}} \begin{bmatrix} \mathbf{S}_1 \\ \vdots \\ \mathbf{S}_\zeta \end{bmatrix},$$

where each $\mathbf{S}_i \in \mathbb{R}^{d/\zeta \times m}$ is an independent CountSketch matrix.

Theorem (Chenakkod, Dereziński, and Dong 2025):[3] Subspace embedding with distortion η if $d = O(n/\eta^2)$ and $\zeta = O(\log(n)/\eta)$.

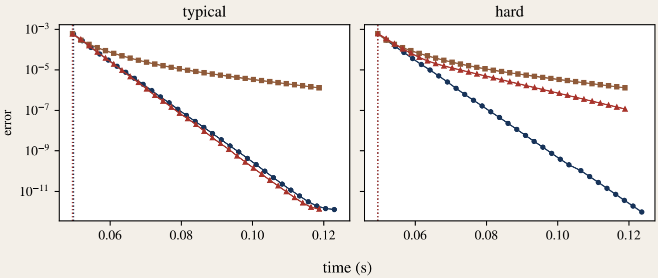
Question: Why did sparse sign sketches get popularized instead of sparse stack sketches? Are there any settings where we should prefer sparse sign sketches?

[3] up to sub-polylog factors.

Other iterative methods

We often have a good idea of the spectrum of **AM**.

- We can use Chebyshev iteration and get essentially the same convergence rate as LSQR, while avoiding inner products.
- However, on “hard problems”, the sketch may not behave like a Gaussian



Legend: Error as a function of runtime for various iterative methods: LSQR ( ), Gradient Descent ( ), and Heavy ball momentum ( ).

Dedicated sketch-apply primitives?

We are using cuSPARSE to apply the sparse sign sketches as generic CSC matrices. However, sparse sketch matrices have a lot of special structure.

- There are only two nonzero values (which we can assume are ± 1). So we can apply the sketch without float multiplication/division.
- Since the nonzeros are distributed in a very particular way, might be more efficient ways to access memory.

Some recent work on fine-grained implementation for CountSketch ($\zeta = 1$).[4]

[4] Higgins, Boman, and Yamazaki 2025.

References I

Avron, Haim, Petar Maymounkov, and Sivan Toledo (Jan. 2010). “Blendenpik: Supercharging LAPACK’s Least-Squares Solver”. In: SIAM Journal on Scientific Computing 32.3, pp. 1217–1236. ISSN: 1095-7197. DOI: 10.1137/090767911.

Chenakkod, Shabarish, Michał Dereziński, and Xiaoyu Dong (2025). Optimal Subspace Embeddings: Resolving Nelson-Nguyen Conjecture Up to Sub-Polylogarithmic Factors. arXiv: 2508.14234 [cs.DS].

Cohen, Michael B. (Dec. 2015). “Nearly Tight Oblivious Subspace Embeddings by Trace Inequalities”. In: Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms. Society for Industrial and Applied Mathematics, pp. 278–287. DOI: 10.1137/1.9781611974331.ch21.

Epperly, Ethan N. (Oct. 2024). “Fast and Forward Stable Randomized Algorithms for Linear Least-Squares Problems”. In: SIAM Journal on Matrix Analysis and Applications 45.4, pp. 1782–1804. ISSN: 1095-7162. DOI: 10.1137/23m1616790.

References II

Higgins, Andrew J., Erik G. Boman, and Ichitaro Yamazaki (2025). A High Performance GPU CountSketch Implementation and Its Application to Multisketching and Least Squares Problems. arXiv: 2508.14209 [math.NA].

Murray, Riley et al. (2023). Randomized Numerical Linear Algebra : A Perspective on the Field With an Eye to Software. arXiv: 2302.11474 [math.NA].

Rokhlin, Vladimir and Mark Tygert (Sept. 2008). “A fast randomized algorithm for overdetermined linear least-squares regression”. In: Proceedings of the National Academy of Sciences 105.36, pp. 13212–13217. ISSN: 1091-6490. DOI: 10.1073/pnas.0804869105.