Adaptive Sketches for Robust Regression with Importance Sampling

Sepideh Mahabadi
Microsoft Research, Redmond, WA, USA

David P. Woodruff
Carnegie Mellon University, Pittsburgh, PA, USA

Samson Zhou
Carnegie Mellon University, Pittsburgh, PA, USA

Abstract

We introduce data structures for solving robust regression through stochastic gradient descent (SGD) by sampling gradients with probability proportional to their norm, i.e., importance sampling. Although SGD is widely used for large scale machine learning, it is well-known for possibly experiencing slow convergence rates due to the high variance from uniform sampling. On the other hand, importance sampling can significantly decrease the variance but is usually difficult to implement because computing the sampling probabilities requires additional passes over the data, in which case standard gradient descent (GD) could be used instead. In this paper, we introduce an algorithm that approximately samples T gradients of dimension d from nearly the optimal importance sampling distribution for a robust regression problem over n rows. Thus our algorithm effectively runs T steps of SGD with importance sampling while using sublinear space and just making a single pass over the data. Our techniques also extend to performing importance sampling for second-order optimization.

2012 ACM Subject Classification Theory of computation → Streaming, sublinear and near linear time algorithms

Keywords and phrases Streaming algorithms, stochastic optimization, importance sampling

Digital Object Identifier 10.4230/LIPIcs.APPROX/RANDOM.2022.31

Category RANDOM

Related Version *Full Version*: `https://arxiv.org/pdf/2207.07822.pdf`

Funding David P. Woodruff and Samson Zhou were supported by a Simons Investigator Award and by the National Science Foundation under Grant No. CCF-1815840.

1 Introduction

Given a matrix $\mathbf{A} \in \mathbb{R}^{n\times d}$ with rows $\mathbf{a}_1, \ldots, \mathbf{a}_n \in \mathbb{R}^d$ and a measurement/label vector $\mathbf{b} \in \mathbb{R}^n$, we consider the standard regression problem

$$\min_{\mathbf{x}\in\mathbb{R}^d} \mathcal{L}(\mathbf{x}) := \sum_{i=1}^{n} M(\langle \mathbf{a}_i, \mathbf{x}\rangle - b_i),$$

where $M : \mathbb{R} \to \mathbb{R}^{\geq 0}$ is a function, called a measure function, that satisfies $M(x) = M(-x)$ and is non-decreasing in $|x|$. An *M-estimator* is a solution to this minimization problem and for appropriate choices of M, can combine the low variance of L_2 regression with the robustness of L_1 regression against outliers.

The Huber norm, for example, is defined using the measure function $H(x) = \frac{x^2}{2\tau}$ for $|x| \leq \tau$ and $H(x) = |x| - \frac{\tau}{2}$ for $|x| > \tau$, where τ is a threshold that governs the interpolation between L_2 loss for small $|x|$ and L_1 loss for large $|x|$. Indeed, it can often be more reasonable to have robust treatment of large residuals due to outliers or errors and Gaussian treatment of

small residuals [12]. Thus the Huber norm is especially popular and "recommended for almost all situations" [39], because it is the "most robust" [13] due to "the useful computational and statistical properties implied by the convexity and smoothness" [9] of its measure function, which is differentiable at all points.

Since the measure function H for the Huber norm and more generally, the measure function M for many common measure functions is convex, we can consider the standard convex *finite-sum form* optimization problem $\min_{\mathbf{x}\in\mathbb{R}^d} F(\mathbf{x}) := \frac{1}{n}\sum_{i=1}^{n} f_i(\mathbf{x})$, where $f_1, \ldots, f_n : \mathbb{R}^d \to \mathbb{R}$ is a sequence of convex functions that commonly represent loss functions. Whereas gradient descent (GD) performs the update rule $\mathbf{x}_{t+1} = \mathbf{x}_t - \eta_t \nabla F(\mathbf{x}_t)$ on the iterate $\mathbf{x}_t$ at iterations $t = 1, 2, \ldots, T$, stochastic gradient descent (SGD) [32, 27, 26] picks $i_t \in [n]$ in iteration t with probability p_{i_t} and performs the update rule $\mathbf{x}_{t+1} = \mathbf{x}_t - \frac{\eta_t}{np_{i_t}} \nabla f_{i_t}(\mathbf{x}_t)$, where ∇f_{i_t} is the gradient (or a subgradient) of f_{i_t} and η_t is some predetermined learning rate. Effectively, training example i_t is sampled with probability p_{i_t} and the model parameters are updated using the selected example. The SGD update rule only requires the computation of a single gradient at each iteration and provides an unbiased estimator to the full gradient, compared to GD, which evaluates n individual gradients in each iteration and is prohibitively expensive for large n. However, since SGD is often performed with uniform sampling, so that the probability $p_{i,t}$[1] of choosing index $i \in [n]$ at iteration t is $p_{i,t} = \frac{1}{n}$ at all times, the variance introduced by the randomness of sampling a specific vector function can be a bottleneck for the convergence rate of this iterative process. Thus, the subject of variance reduction beyond uniform sampling has been well-studied in recent years [33, 18, 11, 31, 40, 10, 25, 35, 19, 21, 34, 30].

A common technique to reduce variance is importance sampling, where the probabilities $p_{i,t}$ are chosen so that vector functions with larger gradients are more likely to be sampled. One such setting of importance sampling is to set the probability of sampling a gradient with probability proportional to its L_2 norm, so that

$$p_{i,t} = \frac{\|\nabla f_i(\mathbf{x}_t)\|_2}{\sum_{j\in[t]} \|\nabla f_j(\mathbf{x}_t)\|_2}.$$

Under these sampling probabilities, importance sampling gives variance

$$\sigma_{opt,t}^2 = \frac{1}{n^2}\left(\left(\sum_{i=1}^{n} \|\nabla f_i(\mathbf{x}_t)\|_2\right)^2 - n^2 \cdot \|\nabla F(\mathbf{x}_t)\|_2^2\right),$$

where we define the variance of a random vector $\mathbf{v}$ to be $\mathrm{Var}(\mathbf{v}) := \mathbb{E}\left[\|\mathbf{v}\|_2^2\right] - \|\mathbb{E}[\mathbf{v}]\|_2^2$, and we define $\sigma_{opt,t}^2$ to be the variance of the random vector $\mathbf{v}$ produced at time t by importance sampling.

By comparison, the probabilities for uniform sampling $p_{i,t} = \frac{1}{n}$ imply $\sigma_t^2 = \mathrm{Var}\left(\frac{1}{np_{i_t,t}}\right)$ and thus the variance $\sigma_{uni,t}^2$ for uniform sampling satisfies

$$\sigma_{uni,t}^2 = \frac{1}{n^2}\left(n\sum_{i=1}^{n} \|\nabla f_i(\mathbf{x}_t)\|_2^2 - n^2 \cdot \|\nabla F(\mathbf{x}_t)\|_2^2\right).$$

By the root mean square-arithmetic mean inequality, the variance of importance sampling is always at most the variance of uniform sampling, and can be significantly less. Hence $\sigma_{opt,t}^2 \leq \sigma_{uni,t}^2$, so that the variance at each step is reduced, possibly substantially, by performing importance sampling instead of uniform sampling.

[1] In contrast to $p_{i,t}$, the term p_{i_t} denotes the probability associated with the specific index i_t chosen at time t.

To see examples where uniform sampling an index performs significantly worse than importance sampling, consider $\nabla f_i(\mathbf{x}) = \langle \mathbf{a}_i, \mathbf{x} \rangle \cdot \mathbf{a}_i$. Then for $\mathbf{A} = \mathbf{a}_1 \circ \ldots \circ \mathbf{a}_n$:

▶ **Example 1.** When the non-zero entries of the input $\mathbf{A}$ are concentrated in a small number of vectors $\mathbf{a}_i$, uniform sampling will frequently sample gradients that are small and make little progress, whereas importance sampling will rarely do so. In an extreme case, the input $\mathbf{A}$ can contain exactly one non-zero vector $\mathbf{a}_i$ and importance sampling will always output the full gradient, whereas uniform sampling will only find the non-zero row with probability $\frac{1}{n}$, so that $\sigma^2_{uni,t} = n \cdot \sigma^2_{opt,t}$.

▶ **Example 2.** It may be that all rows of $\mathbf{A}$ have large magnitude, but $\mathbf{x}$ is nearly orthogonal to most of the rows of $\mathbf{A}$, but is well-aligned with row $\mathbf{a}_r$. Then $\langle \mathbf{a}_i, \mathbf{x} \rangle \cdot \mathbf{a}_i$ is small in magnitude for most i, but $\langle \mathbf{a}_r, \mathbf{x} \rangle \cdot \mathbf{a}_r$ is large so uniform sampling will often output small gradients while importance sampling will output $\langle \mathbf{a}_r, \mathbf{x} \rangle \cdot \mathbf{a}_r$ with high probability, so that it can be that $\sigma^2_{uni,t} = \Omega(n) \cdot \sigma^2_{opt,t}$.

▶ **Example 3.** More generally for a parameter $\nu \in [0, 1]$, if a ν-fraction of the n gradients lengths are bounded by $\mathcal{O}(n)$ while the other $1 - \nu$ fraction of the n gradient lengths are bounded by $\text{poly}(d) \ll n$, then the variance for uniform sampling satisfies $\sigma^2_{uni,t} = \mathcal{O}\left(\nu n^2\right) + \text{poly}(d)$ while the variance for importance sampling satisfies $\mathcal{O}\left(\nu^2 n^2\right) + \text{poly}(d)$.

In fact, it follows from the Cauchy-Schwarz inequality that the importance sampling probability distribution is the *optimal* distribution for variance reduction.

However, computing the probability distribution for importance sampling requires computing the gradients in each round, which creates a "chicken and egg" problem because computing the gradients is too expensive in the first place, or else it is feasible to just run gradient descent. Unfortunately, computing the sampling probabilities in each iteration often requires additional passes over the data, e.g., to compute the gradients in each step, which is generally prohibitively expensive. This problem often prevents importance sampling from being widely deployed.

In this paper, we overcome this problem by introducing efficient sketches for a wide range of M-estimators that can enable importance sampling *without* additional passes over the data. Using our sketches for various measure functions, we give a time-efficient algorithm that *provably* approximates the optimal importance sampling distribution within a constant factor. Thus we can surprisingly simulate T steps of SGD with (nearly) the optimal sampling distribution, while only using a single pass over the data, which avoids the aforementioned problem.

▶ **Theorem 4.** *Given an input matrix $\mathbf{A} \in \mathbb{R}^{n \times d}$ whose rows arrive sequentially in a data stream along with the corresponding labels of a measurement vector $\mathbf{b} \in \mathbb{R}^d$, and a measure function M whose derivative is a continuous union of piecewise constant or linear functions, there exists an algorithm that performs T steps of SGD with variance within a constant factor of the optimal sampling distribution. The algorithm uses $\tilde{\mathcal{O}}\left(nd^2 + Td^2\right)$ pre-processing time and Td^2 $\text{polylog}(Tnd)$ words of space.*

For T iterations, both GD and optimal importance sampling SGD require T passes over the data, while our algorithm only requires a single pass over the data and uses sublinear space for $nd \gg Td^2$. We remark that although the number T of iterations for SGD may be large, a major advantage of GD and SGD with importance sampling is a significantly smaller number of iterations than SGD with uniform sampling, e.g., as in Example 1 and Example 2, so we should expect $n \gg T$. In particular from known results about the convergence of SGD,

e.g., see Theorem 8, if the diameter of the search space and the gradient lengths $\|\nabla f_i(x_t)\|_2$ are both bounded by $\text{poly}(d)$, then we should expect $T \propto \text{poly}(d)$ even for uniform sampling. More generally, if νn of the gradients have lengths $\Theta(n)$, while the remaining gradients have lengths $\text{poly}(d) \ll n$, then Example 3 and Theorem 8 show that the number of steps necessary for convergence for uniform sampling satisfies $T \propto \mathcal{O}\left(\nu^2 n^4\right) + \text{poly}(d)$, while the number of steps necessary for convergence for importance sampling satisfies $T \propto \mathcal{O}\left(\nu^4 n^4\right)$. Thus for $\nu n = \mathcal{O}\left(n^C\right)$ for $C < 1$, i.e., a sublinear number of gradients have lengths that exceed the input size, we have $T = \mathcal{O}\left(n^{4C}\right)$ and hence for $C < \frac{1}{4}$, we have roughly $T = o(n)$ steps are necessary for convergence for SGD with importance sampling.

Finally, we show in the full version of the paper that our techniques can also be generalized to perform importance sampling for second-order optimization.

1.1 Our Techniques

In addition to our main conceptual contribution that optimal convergence rate of importance sampling for SGD can surprisingly be achieved (up to constant factors) without the "chicken and egg" problem of separately computing the sampling probabilities, we present a number of technical contributions that may be of independent interest. Our first observation is that if we were only running a single step of importance sampling for SGD, then we just want a subroutine that outputs a gradient $G(\langle \mathbf{a}_i, \mathbf{x}\rangle - b_i, \mathbf{a}_i)$ with probability proportional to its norm $\|G(\langle \mathbf{a}_i, \mathbf{x}\rangle - b_i, \mathbf{a}_i)\|_2$.

***G*-sampler.** In particular, we need an algorithm that reads a matrix $\mathbf{A} = \mathbf{a}_1 \circ \ldots \circ \mathbf{a}_n \in \mathbb{R}^{n\times d}$ and a vector $\mathbf{x} \in \mathbb{R}^d$ given *after processing* the matrix $\mathbf{A}$, and outputs (a rough approximation to) a gradient $G(\langle \mathbf{a}_i, \mathbf{x}\rangle - b_i, \mathbf{a}_i)$ with probability roughly

$$\frac{\|G(\langle \mathbf{a}_i, \mathbf{x}\rangle - b_i, \mathbf{a}_i)\|_2}{\sum_{j=1}^n \|G(\langle \mathbf{a}_j, \mathbf{x}\rangle - b_j, \mathbf{a}_j)\|_2}.$$

We call such an algorithm a *G-sampler* and introduce such a single-pass, memory-efficient sampler with the following guarantees:

▶ **Theorem 5.** *Given an (α, ε)-smooth gradient G, there exists an algorithm* Sampler *that outputs a noisy vector $\mathbf{v}$ such that $\|\mathbf{v} - \mathbf{a}_i(\langle \mathbf{a}_i, x\rangle - b_i)\|_2 \le \alpha\|\mathbf{a}_i(\langle \mathbf{a}_i, x\rangle - b_i)|\|_2$ and $\mathbb{E}[\mathbf{v}] = \mathbf{a}_i(\langle \mathbf{a}_i, x\rangle - b_i)$ is $(1 \pm \mathcal{O}(\varepsilon))\frac{\|G(\langle \mathbf{a}_i, \mathbf{x}\rangle - b_i, \mathbf{a}_i)\|_2}{\sum_{j\in[n]} \|G(\langle \mathbf{a}_j, \mathbf{x}\rangle - b_j, \mathbf{a}_j)\|_2} + \frac{1}{\text{poly}(n)}$. The algorithm uses d^2 poly $\left(\log(nT), \frac{1}{\alpha}\right)$ update time per arriving row and Td^2 poly $\left(\log(nT), \frac{1}{\alpha}\right)$ total bits of space.*

We say a gradient G is (α, ε)-smooth if a vector $\mathbf{u}$ that satisfies $\|\mathbf{u} - \mathbf{v}\|_2 \le \alpha\|\mathbf{v}\|_2$ implies that $(1-\varepsilon)\|G(\mathbf{v})\|_2 \le \|G(\mathbf{u})\|_2 \le (1+\varepsilon)\|G(\mathbf{v})\|_2$. In particular, the measure functions discussed in Section 1.2 have gradients that are $(\mathcal{O}(\varepsilon), \varepsilon)$-smooth. For example, the subgradient of the Huber estimator is $\mathbf{a}_i \cdot \text{sgn}(\langle \mathbf{a}_i, \mathbf{x}\rangle - b_i)$ for $|\langle \mathbf{a}_i, \mathbf{x}\rangle - b_i| > \tau$, which may change sign when $\langle \mathbf{a}_i, \mathbf{x}\rangle - b_i$ is close to zero, but its norm will remain the same. Moreover, the form $G(\langle \mathbf{a}_i, \mathbf{x}\rangle - b_i, \mathbf{a}_i)$ necessitates that the gradient can be computed strictly from the two quantities $\langle \mathbf{a}_i, \mathbf{x}\rangle - b_i$ and $\mathbf{a}_i$. Thus Theorem 5 implies that our algorithm can also compute a noisy vector $\mathbf{v}'$ such that $\|\mathbf{v}' - G(\langle \mathbf{a}_i, \mathbf{x}\rangle - b_i, \mathbf{a}_i)\|_2 \le \varepsilon\|G(\langle \mathbf{a}_i, \mathbf{x}\rangle - b_i, \mathbf{a}_i)\|_2$.

Observe that an instance of Sampler in Theorem 5 can be used to simulate a single step of SGD with importance sampling and thus T independent instances of Sampler provide an oracle for T steps of SGD with importance sampling. However, this naïve implementation

does not suffice for our purposes because the overall runtime would be $\tilde{\mathcal{O}}\left(Tnd^2\right)$ so it would be more efficient to just run T iterations of GD. Nevertheless, our G-sampler is a crucial subroutine towards our final algorithm and we briefly describe it here.

An alternative definition of G-sampler is given in [16]. In their setting, the goal is to sample a coordinate $i \in [n]$ of a frequency vector f with probability proportional to $G(f_i)$, where G in their notation is a measure function rather than a gradient. However, because the G-sampler of [16] is not a linear sketch, their approach cannot be easily generalized to our setting where the sampling probability of each row $\mathbf{a}_i$ is a function of $\langle \mathbf{a}_i, \mathbf{x} \rangle$, but the vector $\mathbf{x}$ arrives after the stream is already processed.

Furthermore, because the loss function f may not be scale-invariant, then we should also not expect its gradient to be scale invariant at any location $\mathbf{x} \in \mathbb{R}^d$, i.e., $\nabla f(C\mathbf{x}) \neq C^p \cdot \nabla f(\mathbf{x})$ for any constants $p, C > 0$. Hence, our subroutine Sampler cannot use the standard L_p sampler framework used in [20, 2, 15, 22], which generally rescales each row of $\mathbf{a}_i$ by the inverse of a uniform or exponential random variable. A somewhat less common design for L_p samplers is a level set and subsampling approach [24, 17], due to their suboptimal dependencies on the accuracy parameter ε. Fortunately, because we require $\varepsilon = \mathcal{O}(1)$ to achieve a constant factor approximation, we can use the level set and subsampling paradigm as a starting point for our algorithm. Because the algorithms of [24, 17] only sample entries of a vector implicitly defined from a data stream, our G-sampler construction must (1) sample rows of a matrix implicitly defined from a data stream and (2) permit updates to the sampling probabilities implicitly defined through multiplication of each row $\mathbf{a}_i$ with a vector $\mathbf{x}$ that only arrives after the stream is processed.

G-sampler through level sets and subsampling. To illustrate our method and simplify presentation here, we consider L_2 regression with gradient $\mathbf{A}_i\mathbf{x} := \langle \mathbf{a}_i, \mathbf{x} \rangle \cdot \mathbf{a}_i$, by folding in the measurement vector $\mathbf{b}$ into a column of $\mathbf{A}$ – our full algorithm in Section 2 handles both sampling distributions defined with respect to the norm of a general gradient G in the form of Theorem 5, as well as an independent measurement vector $\mathbf{b}$.

We first partition the rows of $\mathbf{A}$ into separate geometrically growing classes based on their L_2 norms, so that for instance, class C_k contains the rows $\mathbf{a}_i$ of $\mathbf{A}$ such that $2^k \leq \|\mathbf{a}_i\|_2 < 2^{k+1}$. We build a separate data structure for each class C_k, which resembles the framework for L_p norm estimation [14]. We would like to use the approximate contributions of the level sets $\Gamma_1, \ldots, \Gamma_K$, with $K = \mathcal{O}\left(\frac{\log n}{\alpha}\right)$, toward the total mass $F_2(S) = \sum_{i=1}^n \|\mathbf{A}_i\mathbf{x}\|_2$, where a level set Γ_j is informally the set of rows $\mathbf{a}_i$ with $\|\mathbf{A}_i\mathbf{x}\|_2 \in \left[\frac{F_2(S)}{(1+\alpha)^{j-1}}, \frac{F_2(S)}{(1+\alpha)^j}\right]$ and the contribution of a level set Γ_j is $\sum_{i \in \Gamma_j} \|\mathbf{A}_i\mathbf{x}\|_2$. Then we could first sample a level set Γ_j from a class C_k and then uniformly select a row $\mathbf{a}_i$ among those in Γ_j. Indeed, we can run a generalized version of the L_2 heavy-hitter algorithm CountSketch [7] on the stream S to identify the level set Γ_1, since its rows will be heavy with respect to $F_2(S)$. However, the rows of the level sets Γ_j for large j may not be detected by CountSketch. Thus, we create $L = \mathcal{O}(\log n)$ substreams $S_1, \ldots, S_L$, so that substream S_ℓ samples each row of $\mathbf{A}$ with probability $2^{-\ell+1}$, and run an instance of CountSketch on each substream S_ℓ to detect the rows of each level set and thus estimate the contribution of each level set.

Sampling from level sets with small contribution. However, there is still an issue – some level sets have contribution that is too small to well-approximate with small variance. For example, if there is a single row with contribution $\frac{F_2(S)}{(1+\alpha)^j}$, then it might not survive the subsampling at a level S_ℓ that is used to detect it, in which case it will never be sampled. Alternatively, if it is sampled, it will be rescaled by a large amount, so that its level set will

be sampled with abnormally large probability. Instead of handling this large variance, we instead add a number of dummy rows to each level set, to ensure that their contributions are all "significant" and thus be well-approximated.

Now we have "good" approximations to the contributions of each level set within a class, so we can first select a level set with probability proportional to the approximate contributions of each level set and then uniformly sample a row from the level set. Of course, we may uniformly sample a dummy row, in which case we say the algorithm fails to acquire a sample. We show that the contribution added by the dummy rows is a constant fraction, so this only happens with a constant probability. Thus with $\mathcal{O}\left(\log \frac{1}{\delta}\right)$ constant number of independent samples, we can boost the probability of successfully acquiring a sample to $1-\delta$ for any $\delta \in (0,1]$. We then set $\delta = \frac{1}{\text{poly}(n,T,d)}$.

Unbiased samples. Unfortunately, CountSketch using $\mathcal{O}\left(\frac{1}{\alpha^2}\right)$ buckets only guarantees additive $\alpha\, L_2(S)$ error to a particular row with constant probability. To achieve the standard "for-all" guarantee across all n rows, an estimate for each row $\mathbf{a}_i$ is then output by taking the row with the median length across $\mathcal{O}(\log n)$ independent instances. However, the median row is no longer unbiased, which could potentially affect the convergence guarantees of SGD. Instead, we use d separate instances of CountSketch, so that each instance handles a separate coordinate of the vector. Thus if the goal is to output a noisy estimate to $\mathbf{a}_i$, we have a separate CountSketch report each coordinate $(\mathbf{a}_i)_j$, where $j \in [d]$. It can be shown that the median of each estimated coordinate is an unbiased estimate to the true value $(\mathbf{a}_i)_j$ of the coordinate because the probability mass function is symmetric about the true value for each coordinate. Moreover, the error to a single coordinate $(\mathbf{a}_i)_j$ may be large relative to the value of the coordinate in the case that $(\mathbf{a}_i)_j$ is not heavy with respect to $\{(\mathbf{a}_i)_j\}_{i\in[n]}$. However, we show that the "overall" error to all coordinates of $\mathbf{a}_i$ is small relative to $\|\mathbf{a}_i\|_2$, due to $\mathbf{a}_i$ being a "heavy" row at the appropriate subsampling level.

Stochastic gradient descent with importance sampling. The main problem with the proposed G-sampler is that it requires reading the entire matrix $\mathbf{A}$ but it cannot be repeatedly used without incurring dependency issues. In particular, if a sampler at the first iteration of SGD outputs a gradient $\mathbf{A}_{i_1}\mathbf{x}_1$ that is used to construct $\mathbf{x}_2$, then $\mathbf{x}_2$ is not independent of the sampler and thus the same sampler should not be used to sample $\mathbf{A}_{i_2}\mathbf{x}_2$. This suggests that if we want to perform T steps of SGD with importance sampling, then we would require T separate data structures, which would require Tnd time to construct for dense matrices, but then we might as well just perform full gradient descent!

Instead in Section 3, we partition the matrix $\mathbf{A}$ among multiple buckets and create a sampler for each bucket. Now as long as each bucket should have been sampled a single time, then we will have a fresh sampler with independent randomness for each time a new bucket is sampled. If we perform T steps of SGD with importance sampling, then roughly T buckets should suffice, but we cannot guarantee that each bucket is sampled a single time. For example, if only a single $\mathbf{A}_i$ is non-zero, then whichever bucket $\mathbf{A}_i$ is assigned to will be sampled every single time.

Now the challenge is identifying the submatrices $\mathbf{A}_i = \mathbf{a}_i^\top \mathbf{a}_i$ that may be sampled multiple times, since we do not know the values of the vectors $\mathbf{x}_1, \ldots, \mathbf{x}_T$ a priori. Fortunately, we know that $\|\mathbf{A}_i\mathbf{x}_t\|_2$ can only be large if $\mathbf{a}_i$ has high sensitivity, where we define the sensitivity for a row $\mathbf{a}_i$ in $\mathbf{A}$ to be the quantity $\max_{\mathbf{x}\in\mathbb{R}^d} \frac{\left\|\mathbf{a}_i^\top(\langle \mathbf{a}_i,\mathbf{x}\rangle)\right\|_2}{\sum_{j=1}^n \left\|\mathbf{a}_j^\top(\langle \mathbf{a}_j,\mathbf{x}\rangle)\right\|_2}$. Thus if a block is sampled multiple times, then one of its rows must have large sensitivity.

Hence, we would like to identify the buckets that contain any row with sensitivity at least $\frac{1}{T}$ and create T independent samplers for those buckets so that even if the same bucket is sampled all T times, there will be a fresh sampler available. Crucially, the process of building separate buckets for the rows with the large sensitivities can be identified in just a single pass over the data.

We remark that since each row has sensitivity $\max_{\mathbf{x}\in\mathbb{R}^d} \frac{\left\|\mathbf{a}_i^\top(\langle\mathbf{a}_i,\mathbf{x}\rangle)\right\|_2}{\sum_{j=1}^n \left\|\mathbf{a}_j^\top(\langle\mathbf{a}_j,\mathbf{x}\rangle)\right\|_2}$, then it can be shown that the sum of the sensitivities is $\mathcal{O}(d\log n)$ by partitioning the rows into $\mathcal{O}(\log n)$ classes $C_1, C_2, \ldots$ of exponentially increasing norm, so that $\mathbf{a}_i \in C_\ell$ if $2^\ell \le \|\mathbf{a}_i\|_2 < 2^{\ell+1}$. We then note that the sensitivity of each row $\mathbf{a}_i \in C_\ell$ is upper bounded by $\max_{\mathbf{x}\in\mathbb{R}^d} \frac{|\langle\mathbf{a}_i,\mathbf{x}\rangle|}{\sum_{\mathbf{a}_j\in C_\ell}|\langle\mathbf{a}_j,\mathbf{x}\rangle|}$. However, this latter quantity is an L_1 sensitivity, whose sum is known to be bounded by $\mathcal{O}(d)$, e.g., [8]. Thus the sum of the sensitivities in each class is at most $\mathcal{O}(d)$ and so for a matrix $\mathbf{A}$ whose entries are polynomially bounded by n, the sum of the sensitivities is at most $\mathcal{O}(d\log n)$.

Unfortunately, since the sensitivities sum to $\mathcal{O}(d\log n)$, there can be up to Td rows with sensitivity at least $\frac{1}{T}$, so creating T independent samplers corresponding to each of these rows would yield $\Omega(T^2 d)$ samplers, which is a prohibitive amount of space. Instead, we simply remove the rows with large sensitivities from the buckets and store them explicitly. We then show this approach still avoids any sampler from being used multiple times across the T iterations while also enabling the data structure to just use $\tilde{\mathcal{O}}(Td)$ samplers. Now since we can explicitly consider the rows with sensitivities roughly at least $\frac{1}{T}$, then we can use $\Theta(T)$ buckets in total to ensure that the remaining non-zero entries of $\mathbf{A}$ are partitioned evenly across buckets that will only require $\Theta(\log(Td))$ independent samplers.

1.2 Applications

In this section, we discuss applications of our result to commonly used loss functions, such as L_p loss or various M-estimators, e.g., [9, 8, 37, 29].

***L_1* and *L_2* regression.** The L_p regression loss function is defined using $f_i(\mathbf{x}) = |\mathbf{a}_i^\top\mathbf{x} - b_i|^p$. The case $p = 2$ corresponds to the standard least squares regression problem, while $p = 1$ corresponds to least absolute deviation regression, which is more robust to outliers than least squares, but also less stable and with possibly multiple solutions. For $p = 1$, the subgradient is $\mathbf{a}_i \cdot \operatorname{sgn}(\langle\mathbf{a}_i,\mathbf{x}\rangle - b_i)$ while for $p = 2$, the subgradient is $2\mathbf{a}_i(\langle\mathbf{a}_i,\mathbf{x}\rangle - b_i)$.

Huber estimator. As previously discussed, Huber loss [13] is commonly used, e.g., [39, 9], to achieve Gaussian properties for small residuals [12] and robust properties for large residuals due to outliers or errors. The Huber estimator is also within a constant factor of other M-estimators that utilize the advantage of the L_1 loss function to minimize the impact of large errors/outliers and that of the L_2 loss function to be convex, such as the L_1-L_2 estimator and the Fair estimator [4]. Given a threshold $\tau > 0$, the Huber loss H is defined by $H(x) = \frac{x^2}{2\tau}$ for $|x| \le \tau$ and $H(x) = |x| - \frac{\tau}{2}$ for $|x| > \tau$. Thus the subgradient for H is $\frac{\mathbf{a}_i}{\tau}(\langle\mathbf{a}_i,\mathbf{x}\rangle - b_i)$ for $|\langle\mathbf{a}_i,\mathbf{x}\rangle - b_i| \le \tau$ and $\mathbf{a}_i \cdot \operatorname{sgn}(\langle\mathbf{a}_i,\mathbf{x}\rangle - b_i)$ for $|\langle\mathbf{a}_i,\mathbf{x}\rangle - b_i| > \tau$.

Ridge regression. It is often desirable for a solution $\mathbf{x}$ to be sparse. The natural approach to encourage sparse solutions is to add a regularization $\lambda\|\mathbf{x}\|_0$ term to the loss function, for some parameter $\lambda > 0$. However, since $\|\mathbf{x}\|_0$ is not convex, ridge regression is often used as a convex relaxation that encourage sparse solutions. The ridge regression loss function satisfies

$f_i(\mathbf{x}) = (\mathbf{a}_i^\top \mathbf{x} - b_i)^2 + \lambda \|\mathbf{x}\|_2^2$ for each $i \in [n]$, so that λ regularizes the penalty term associated with the squared magnitude of $\mathbf{x}$. Higher values of λ push the optimal solution towards zero, which leads to lower variance, as a particular coordinate has a smaller effect on the prediction. The gradient for the ridge regression loss function satisfies $\nabla f_i(\mathbf{x}) = 2\mathbf{a}_i(\langle \mathbf{a}_i, \mathbf{x}\rangle - b_i) + 2\lambda\mathbf{x}$.

Lasso. Another approach that encourages sparsity is using the L_1 regularization instead of the L_2 regularization. Least absolute shrinkage and selection operator (Lasso) regression uses the loss function $f_i(\mathbf{x}) = (\mathbf{a}_i^\top \mathbf{x} - b_i)^2 + \lambda \|\mathbf{x}\|_1$. Whereas the penalty term associated for ridge regression will drive down the Euclidean norm of $\mathbf{x}$ for larger λ, solutions with large L_1 norm are still possible if the mass of $\mathbf{x}$ is spread across a large number of coordinates. By contrast, the penalty term associated for Lasso drives down the total magnitude of the coordinates of $\mathbf{x}$. Thus, in this sense, Lasso tends to drive coordinates to zero and encourages sparsity, which does not usually happen for ridge regression. The subgradient for the Lasso regression loss function satisfies $\nabla f_i(\mathbf{x}) = 2\mathbf{a}_i(\langle \mathbf{a}_i, \mathbf{x} - b_i) + 2\lambda \operatorname{sgn}(\mathbf{x})$, where we abuse notation by using $\operatorname{sgn}(\mathbf{x})$ to denote the coordinate-wise sign of the entries of $\mathbf{x}$.

Group lasso. [38] proposed Group Lasso as a generalization to Lasso. Suppose the weights in $\mathbf{x}$ can be grouped into m groups: $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m)}$. We group the columns of $\mathbf{A} = \mathbf{a}_1 \circ \ldots \circ \mathbf{a}_n$ so that $\mathbf{A}^{(i)}$ is the set of columns that corresponds to the weights in $\mathbf{x}^{(i)}$. The Group Lasso function is defined as $f_i(\mathbf{x}) = (\mathbf{a}_i^\top \mathbf{x} - b_i)^2 + \lambda \sum_{j=1}^m \sqrt{G_j} \|\mathbf{x}^{(j)}\|_2$, where G_j represents the number of weights in $\mathbf{x}^{(j)}$. Note that Group Lasso becomes Lasso for $m = n$.

1.3 Preliminaries

For an integer $n > 0$, we use $[n]$ to denote the set $\{1, 2, \ldots, n\}$. We use boldfaced font for variables that represent either vectors of matrices and plain font to denote variables that represent scalars. We use the notation $\tilde{\mathcal{O}}(\cdot)$ to suppress polylog factors, so that $f(T, n, d) = \tilde{\mathcal{O}}(g(T, n, d))$ implies that $f(T, n, d) \leq g(T, n, d) \operatorname{polylog}(Tnd)$. Let $\mathbf{A} \in \mathbb{R}^{n \times d}$ and $\mathbf{B} \in \mathbb{R}^{m \times d}$. We use $\circ$ to denote vertical concatenation, so that $\mathbf{A} \circ \mathbf{B} = \begin{bmatrix} \mathbf{A} \\ \mathbf{B} \end{bmatrix}$, and $\otimes$ to denote outer product, so that the (i, j)-th entry of the matrix $\mathbf{u} \otimes \mathbf{v} \in \mathbb{R}^{m \times n}$ for $\mathbf{u} \in \mathbb{R}^m$ and $\mathbf{v} \in \mathbb{R}^n$ is $u_i v_j$. For a vector $\mathbf{v} \in \mathbb{R}^n$, we let $\|\mathbf{v}\|_p^p = \sum_{i=1}^n v_i^p$ and $\|\mathbf{v}\|_\infty = \max_i |v_i|$. For a matrix $\mathbf{A} \in \mathbb{R}^{n \times d}$, we denote the Frobenius norm of $\mathbf{A}$ by $\|\mathbf{A}\|_F = \sqrt{\sum_{i=1}^n \sum_{j=1}^d A_{i,j}^2}$. We also use $\|\mathbf{A}\|_p = \left(\sum_{i=1}^n \sum_{j=1}^d |A_{i,j}|^p\right)^{\frac{1}{p}}$. For a function f, we use ∇f to denote its gradient.

▶ **Definition 6.** *A function $f : \mathbb{R}^d \to \mathbb{R}$ is* convex *if $f(\mathbf{x}) \geq f(\mathbf{y}) + \langle \nabla f(\mathbf{y}), \mathbf{x} - \mathbf{y}\rangle$ for all $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$.*

▶ **Definition 7.** *A continuously differentiable function $f : \mathbb{R}^d \to \mathbb{R}$ is* μ-smooth *if*

$$\|\nabla f(\mathbf{x}) - \nabla f(\mathbf{y})\|_2 \leq \mu \|\mathbf{x} - \mathbf{y}\|_2 ,$$

for all $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$. Then it follows, e.g., by Lemma 3.4 in [6], that for every $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$,

$$|f(\mathbf{y}) - f(\mathbf{x}) - \langle \nabla f(\mathbf{x}), \mathbf{y} - \mathbf{x}\rangle| \leq \frac{\mu}{2} \|\mathbf{y} - \mathbf{x}\|_2^2 .$$

Recall that SGD offers the following convergence guarantees for smooth convex functions:

▶ **Theorem 8** ([26, 23]). *Let F be a μ-smooth convex function and $\mathbf{x}_{\mathsf{opt}} = \operatorname{argmin} F(\mathbf{x})$. Let σ^2 be an upper bound for the variance of the unbiased estimator across all iterations and $\overline{\mathbf{x}_k} = \frac{\mathbf{x}_1+\ldots+\mathbf{x}_k}{k}$. Let each step-size η_t be $\eta \leq \frac{1}{\mu}$. Then for SGD with initial position $\mathbf{x}_0$, and any value of k,*

$$\mathbb{E}\left[F(\overline{\mathbf{x}_k}) - F(\mathbf{x}_{\mathsf{opt}})\right] \leq \frac{1}{2\eta k}\left\|\mathbf{x}_0 - \mathbf{x}_{\mathsf{opt}}\right\|_2^2 + \frac{\eta\sigma^2}{2}.$$

This means that $k = \mathcal{O}\left(\frac{1}{\varepsilon^2}\left(\sigma^2 + \mu\left\|\mathbf{x}_0 - \mathbf{x}_{\mathsf{opt}}\right\|_2^2\right)^2\right)$ iterations suffice to obtain an ε-approximate optimal value by setting $\eta = \frac{1}{\sqrt{k}}$.

2 G-Sampler Algorithm

In this section, we describe our G-sampler, which reads a matrix $\mathbf{A} = \mathbf{a}_1 \circ \ldots \circ \mathbf{a}_n \in \mathbb{R}^{n\times d}$ and a vector $\mathbf{x} \in \mathbb{R}^d$ given after processing the matrix $\mathbf{A}$, and outputs a gradient $G(\langle \mathbf{a}_i, \mathbf{x}\rangle - b_i, \mathbf{a}_i)$ among the n gradients $\{G(\langle \mathbf{a}_1, \mathbf{x}\rangle - b_1, \mathbf{a}_1), \ldots, G(\langle \mathbf{a}_n, \mathbf{x}\rangle - b_n, \mathbf{a}_n)\}$ with probability roughly $\frac{\|G(\langle \mathbf{a}_i,\mathbf{x}\rangle - b_i, \mathbf{a}_i)\|_2}{\sum_{j=1}^n \|G(\langle \mathbf{a}_j,\mathbf{x}\rangle - b_j, \mathbf{a}_j)\|_2}$. However, it is not possible to exactly return $G(\langle \mathbf{a}_i, \mathbf{x}\rangle - b_i, \mathbf{a}_i)$ using sublinear space; we instead return a vector $\mathbf{v}$ such that $\mathbb{E}[\mathbf{v}] = G(\langle \mathbf{a}_i, \mathbf{x}\rangle - b_i, \mathbf{a}_i)$ and $\|\mathbf{v} - G(\langle \mathbf{a}_i, \mathbf{x}\rangle - b_i, \mathbf{a}_i)\| \leq \varepsilon\|G(\langle \mathbf{a}_i, \mathbf{x}\rangle - b_i, \mathbf{a}_i)\|_2$. To achieve our G-sampler, we first require a generalization of the standard L_2-heavy hitters algorithm CountSketch [7], which we describe in Section 2.1. We then describe our G-sampler in full in Section 2.2.

2.1 Heavy-Hitters

Before describing our generalization of CountSketch, we first require the following F_G estimation algorithm that generalizes both well-known frequency moment estimation algorithm of [1, 36] and symmetric norm estimation algorithm of [3] by leveraging the linear sketches used in those data structures to support "post-processing" with multiplication by any vector $\mathbf{x} \in \mathbb{R}^d$.

▶ **Theorem 9** ([3]). *Given a constant $\varepsilon > 0$ and an (α, ε)-smooth gradient G, there exists a one-pass streaming algorithm Estimator that takes updates to entries of a matrix $\mathbf{A} \in \mathbb{R}^{n\times d}$, as well as vectors $\mathbf{x} \in \mathbb{R}^d$ and $\mathbf{b} \in \mathbb{R}^d$ that arrive after the stream, and outputs a quantity $\hat{F}$ such that $(1-\varepsilon)\sum_{i\in[n]}\|G(\langle \mathbf{a}_i, \mathbf{x}\rangle - b_i, \mathbf{a}_i)\|_2 \leq \hat{F} \leq (1+\varepsilon)\sum_{i\in[n]}\|G(\langle \mathbf{a}_i, \mathbf{x}\rangle - b_i, \mathbf{a}_i)\|_2$. The algorithm uses $\frac{d^2}{\alpha^2}\operatorname{polylog}(nT)$ bits of space and succeeds with probability at least $1 - \frac{1}{\operatorname{poly}(n,T)}$.*

We now describe a straightforward generalization of the L_2-heavy hitter algorithm CountSketch so that (1) it can find the "heavy rows" of a matrix $\mathbf{A} = \mathbf{a}_1 \circ \ldots \circ \mathbf{a}_n \in \mathbb{R}^{n\times d}$ rather than the "heavy coordinates" of a vector and (2) it supports post-processing multiplication by a vector $\mathbf{x} \in \mathbb{R}^d$ that arrives only after $\mathbf{A}$ is processed. Let $\mathbf{A}_i = \mathbf{a}_i \otimes \mathbf{a}_i \in \mathbb{R}^{d\times d}$ for all $i \in [n]$. We define $\operatorname{tail}(c)$ to be the $n - c$ rows that do not include the top c values of $\|\mathbf{A}_i\mathbf{x}\|_2$. For a given $\varepsilon > 0$, we say a block $\mathbf{A}_i$ with $i \in [n]$ is *heavy* if $\|\mathbf{A}_i\mathbf{x}\|_2 \geq \varepsilon\sum_{i\in\operatorname{tail}(2/\varepsilon^2)}\|\mathbf{A}_i\mathbf{x}\|_2$.

The standard CountSketch algorithm for finding the L_2-heavy hitters among the coordinates of a vector $\mathbf{v}$ of dimension n works by hashing the universe $[n]$ across $\mathcal{O}\left(\frac{1}{\varepsilon^2}\right)$ buckets. Each coordinate $i \in [n]$ is also given a random sign σ_i and so the algorithm maintains the $\mathcal{O}\left(\frac{1}{\varepsilon^2}\right)$ signed sums $\sum\sigma_i x_i$ across all the coordinates hashed to each bucket. Then to estimate x_i, the algorithm simply outputs $\sigma_i C_{h(i)}$, where $C_{h(i)}$ represents the counter corresponding to the bucket to which coordinate i hashes. It can be shown that

$\mathbb{E}\left[\sigma_i C_{h(i)}\right] = x_i$, where the expectation is taken over the random signs σ and the choices of the hash functions. Similarly, the variance of the estimator can be bounded to show that with constant probability, the estimator has additive error $\mathcal{O}(\varepsilon)\|\mathbf{x}\|_2^2$ to x_i with constant probability. Thus if $x_i > \varepsilon\|\mathbf{x}\|_2^2$, the algorithm will be able to identify coordinate i as a heavy-hitter (in part by allowing some false positives). We give the algorithm in full in Algorithm 1.

Algorithm 1 Output heavy vectors $(\langle \mathbf{a}_i, \mathbf{x}\rangle)\mathbf{a}_i$, where $\mathbf{x}$ can be a vector that arrives after $\mathbf{A}$ is processed.

Input: Matrix $\mathbf{A} \in \mathbb{R}^{n\times d}$, vector $\mathbf{x} \in \mathbb{R}^d$, accuracy parameter $\varepsilon > 0$, failure parameter $\delta \in (0, 1]$.

Output: Noisy vectors $\mathbf{a}_i^\top \mathbf{a}_i \mathbf{x}$ with $\left\|\mathbf{a}_i^\top \mathbf{a}_i \mathbf{x}\right\|_2^2 \geq \varepsilon^2 \sum_{i\in \mathrm{tail}(2/\varepsilon^2)} \left\|\mathbf{a}_i^\top \mathbf{a}_i \mathbf{x}\right\|_2^2$.

$b \leftarrow \Omega\left(\frac{1}{\delta\varepsilon^4}\right)$
Let $\mathcal{T}$ contain b buckets, each initialized to the all zeros $\mathbb{R}^{d\times d}$ matrix.
Let $\sigma_i \in \{-1, +1\}$ be drawn from 4-wise independent family for $i \in [n]$.
Let $h : [n] \to [b]$ be 2-wise independent
Process A:
Let $\mathbf{A} = \mathbf{a}_1 \circ \ldots \circ \mathbf{a}_n$, where each $\mathbf{a}_i \in \mathbb{R}^d$.
for each $j = 1$ to n **do**
    $\mathbf{A}_j \leftarrow \mathbf{a}_j \otimes \mathbf{a}_j$
    Add $\sigma_j \mathbf{A}_j$ to the matrix in bucket $h(j)$.
Let $\mathbf{M}_j$ be the matrix in bucket j of $\mathcal{T}$ for $i \in [r], j \in [b]$.
Process x:
for $j \in [b]$ **do**
    $\mathbf{v}_j \leftarrow \mathbf{M}_j \mathbf{x}$
On query $k \in [n]$, report $\sigma_k \mathbf{v}_{h(k)}$.

Thus Algorithm 1 can be used to give the following guarantee by taking the median of the norms of $\mathcal{O}(\log(nT))$ copies, as well as the vector that realizes the median.

▶ **Lemma 10** ([22]). *There exists an algorithm that uses $\mathcal{O}\left(\frac{d^2}{\varepsilon^2}\log^2 n\right)$ space and outputs a set S of indices so that with probability $1 - \frac{1}{\mathrm{poly}(n,T)}$, for all $i \in [n]$, $i \in S$ if $\|\mathbf{A}_i\mathbf{x}\|_2 \geq \varepsilon \sum_{j\in\mathrm{tail}(2/\varepsilon^2)} \|\mathbf{A}_j\mathbf{x}\|_2$ and $i \notin S$ if $\|\mathbf{A}_i\mathbf{x}\|_2 \leq \frac{\varepsilon}{2} \sum_{j\in\mathrm{tail}(2/\varepsilon^2)} \|\mathbf{A}_j\mathbf{x}\|_2$. The algorithm uses $\mathcal{O}\left(\frac{d^2}{\varepsilon^2}\log^2(nT)\right)$ space.*

However, the vector that realizes the median of the norms may no longer be an unbiased estimate to each heavy-hitter. Unfortunately, we shall require unbiased estimates to each heavy-hitter, because we will use estimated heavy-hitters as unbiased gradients as part of SGD with importance sampling. Thus we give an additional algorithm so that for each $i \in S$ reported by Algorithm 1, the algorithm outputs an *unbiased* estimate to the vector $(\langle \mathbf{a}_i, \mathbf{x}\rangle)\mathbf{a}_i$ with a "small" error, in terms of the total mass $\sum_{i\in\mathrm{tail}(2/\varepsilon^2)} \|\mathbf{A}_i\mathbf{x}\|_2$ excluding the largest $\frac{2}{\varepsilon^2}$ rows.

To that end, we instead run d separate instances of CountSketch to handle the d separate coordinates of each heavy-hitter $\mathbf{A}_i\mathbf{x}$. We show that the median of each estimated coordinate is an unbiased estimate to the coordinate $(\mathbf{A}_i\mathbf{x})_j$, since the probability mass function is symmetric about the true value for each coordinate. Furthermore, we show that although the error to a single coordinate $(\mathbf{A}_i\mathbf{x})_j$ may be large compared to $|(\mathbf{A}_i\mathbf{x})_j|$, the error is not large compared to $\sum_{i\in\mathrm{tail}(2/\varepsilon^2)} \|\mathbf{A}_i\mathbf{x}\|_2$.

▶ **Lemma 11.** *There exists an algorithm that uses* $\mathcal{O}\left(\frac{d^2}{\varepsilon^2}\log^2(nT)\right)$ *space and outputs a vector* $\mathbf{y}_i$ *for each index* $i \in [n]$ *so that* $|\,\|\mathbf{y}_i\|_2 - \|\mathbf{A}_i\mathbf{x}\|_2\,| \leq \varepsilon \sum_{i\in \mathrm{tail}(2/\varepsilon^2)} \|\mathbf{A}_i\mathbf{x}\|_2$ *and* $\mathbb{E}[\mathbf{y}_i] = \mathbf{A}_i\mathbf{x}$ *with probability at least* $1 - \frac{1}{\mathrm{poly}(n,T)}$.

However if say, we want to identify the heavy gradients $(\langle \mathbf{a}_i, \mathbf{x}\rangle - b_i)\mathbf{a}_i$, then we create separate data structures for the constant (in $\mathbf{x}$) term $b_i\mathbf{a}_i$ and the linear term $(\mathbf{a}_i \otimes \mathbf{a}_i)\mathbf{x}$, using the same buckets, hash functions, and random signs. For the constant term data structure, we hash the scaled rows $b_i\mathbf{a}_i$ into $\mathcal{O}\left(\frac{1}{\varepsilon^2}\right)$ buckets, so that each bucket contains a vector that represents the signed sum of the (scaled) rows of $\mathbf{A}$ that hash to the bucket. For the linear term data structure, we hash the outer products $\mathbf{A}_i := \mathbf{a}_i \otimes \mathbf{a}_i$ into $\mathcal{O}\left(\frac{1}{\varepsilon^2}\right)$ buckets, so that each bucket contains a vector that represents the signed sum of the matrices $\mathbf{A}_i$ that hash to the bucket. Once the vector $\mathbf{x}$ arrives after $\mathbf{A}$ is processed, then we can multiply each of the matrices stored by each bucket by $\mathbf{x}$. Since the signed sum is a linear sketch, this procedure is equivalent to originally taking the signed sums of the vectors $\mathbf{A}_i\mathbf{x}$. Similarly, by linearity, we can then take any linear combination of the two data structures to identify the heavy gradients $(\langle \mathbf{a}_i, \mathbf{x}\rangle - b_i)\mathbf{a}_i$.

2.2 G-Sampler Algorithm

In this section, we first describe our G-sampler algorithm, where we sample a gradient $G(\langle \mathbf{a}_i, \mathbf{x}\rangle - b_i, \mathbf{a}_i)$ with probability proportional to $\|G(\langle \mathbf{a}_i, \mathbf{x}\rangle - b_i, \mathbf{a}_i)\|_2$. Given an accuracy parameter $\varepsilon > 0$, let α be a constant, parametrized by ε, so that $(1-\varepsilon)F_G(\mathbf{v}) \leq F_G(\mathbf{u}) \leq (1+\varepsilon)F_G(\mathbf{v})$, for any $\mathbf{u}$ with $\|\mathbf{u}-\mathbf{v}\|_2 \leq \alpha\|\mathbf{v}\|_2$. As our data structure will be a linear sketch, we focus on the case where we fold the measurement vector $\mathbf{b}$ into a column of $\mathbf{A}$, so that we want to output a gradient $\mathbf{A}_i\mathbf{x} := (\mathbf{a}_i \otimes \mathbf{a}_i)\mathbf{x}$.

Our algorithm first partitions the rows of $\mathbf{A}$ into classes, based on their L_2 norm. For example, if all entries of $\mathbf{A}$ are integers, then we define class $C_k := \{\mathbf{a}_i : 2^{k-1} \leq \|\mathbf{a}_i\|_2 < 2^k\}$. We create a separate data structure for each class. We will use the F_G estimation algorithm on each class to first sample a particular class. It then remains to sample a particular vector $(\mathbf{a}_i \otimes \mathbf{a}_i)\mathbf{x}$ from a class.

Depending on the vector $\mathbf{x}$, the vectors $(\mathbf{a}_i \otimes \mathbf{a}_i)\mathbf{x}$ in a certain class C_k can have drastically different L_2 norm. We define level set Γ_j as the vectors that satisfy $(1+\varepsilon)^{j-1} \leq \|(\mathbf{a}_i \otimes \mathbf{a}_i)\mathbf{x}\|_2 < (1+\varepsilon)^j$. If we could estimate $|\Gamma_j|$, then we could estimate the contribution of each level set Γ_j toward the overall mass $\sum_{i\in C_k} \|(\mathbf{a}_i \otimes \mathbf{a}_i)\mathbf{x}\|_2$, so we can then sample a specific level set Γ_j from the class C_k. To that end, we create $L = \mathcal{O}(\log n)$ substreams, $S_1, \ldots, S_L$, so that we sample each row with probability $\frac{1}{2^{\ell-1}}$ in substream S_ℓ.

The point is that if the contribution of level set Γ_j is "significant", then there exists a specific substream S_ℓ in which the vectors of Γ_j will be likely detected by the heavy-hitter algorithm we introduced in Section 2.1, if they are sampled by S_ℓ. We can then use these vectors that are output by the heavy-hitter algorithm to estimate the contribution of level set Γ_j. However, if the contribution of level set Γ_j is not significant, then there may not be any vectors of Γ_j that survive the sampling in substream S_ℓ. Thus we add a number of "dummy rows" to each level set to insist that all level sets are significant, so that we can estimate their contributions.

We then sample a level set Γ_j with probability proportional to its contribution and uniformly select a (noisy) vector from the level set. If the selected vector is one of the original rows of the matrix, then we output the noisy vector. Otherwise, we say the sampler has failed. We show that the sampler only fails with constant probability, so it suffices to run $\mathcal{O}\left(\log\frac{1}{\delta}\right)$ independent instances to boost the probability of success to any arbitrary $1-\delta$. The algorithm for selecting a level set Γ_j from a specific class C_k appears in Algorithm 2.

We first show that the dummy rows only contribute at constant multiple of the mass $F_G(S) = \sum_{i=1}^n \|\mathbf{A}_i\mathbf{x}\|_2$, where we assume for simplicity that all rows of $\mathbf{A}$ are in the same class.

▶ **Lemma 12.** *Let S be the input data stream with subsamples $S_1, \ldots, S_L$. Let $\widetilde{S}$ be the input data stream with the additional dummy rows and corresponding subsamples $\widetilde{S_1}, \ldots, \widetilde{S_L}$. Then $2F_G(S) \geq F_G(\widetilde{S}) \geq F_G(S)$.*

We would now like to show that with high probability, each of the substreams have exponentially smaller mass $F_G(S_j)$. However, this may not be true. Consider a single row $\mathbf{a}_i$ that contributes a constant fraction of $F_G(S)$. Then even for $j = \log n$, the probability that $\mathbf{a}_i$ is sampled is roughly $\frac{1}{n} \gg \frac{1}{\text{poly}(n)}$. Instead, we note that CountSketch satisfies the stronger tail guarantee in Lemma 11. Hence for each $j \in [K]$, we define $S_j^{\text{tail}(t)}$ to be the frequency vector S_j with its t largest entries set to zero and we show an exponentially decreasing upper bound on $F_G(\widetilde{S_j^{\text{tail}(t)}})$.

▶ **Lemma 13.** *With high probability, we have that for all $j \in [K]$, $F_G(\widetilde{S_j^{\text{tail}(t)}}) \leq \frac{F_G(S)}{2^j}\log(nT)$ for $t = \mathcal{O}\left(\frac{\log n}{\alpha^3}\right)$.*

We also show that the estimated contribution of each level set (after incorporating the dummy rows) is a $(1+\alpha)$-approximation of the true contribution.

▶ **Lemma 14.** *With high probability, we have that for all $j \in [K]$, $(1-\varepsilon)F_G(\widetilde{S_j}) \leq \widetilde{F_G}(\widetilde{S_j}) \leq (1+\varepsilon)F_G(\widetilde{S_j})$.*

Finally, we show that each row is sampled with the correct distribution and is an unbiased estimate.

▶ **Lemma 15.** *Suppose that $2^k < \|\mathbf{a}_i\|_2 \leq 2^{k+1}$ for all $i \in [n]$. Then the probability that Algorithm 2 outputs a noisy vector $\mathbf{v}$ such that $\|\mathbf{v} - \mathbf{a}_i(\langle \mathbf{a}_i, x\rangle - b_i)\|_2 \leq \alpha\|\mathbf{a}_i(\langle \mathbf{a}_i, x\rangle - b_i)|\|_2$ with $\mathbb{E}[\mathbf{v}] = \mathbf{a}_i(\langle \mathbf{a}_i, x\rangle - b_i)$ is $p_{\mathbf{v}} = (1 \pm \mathcal{O}(\varepsilon))\frac{G(\mathbf{a}_i(\langle \mathbf{a}_i, x\rangle - b_i))}{F_G(\widetilde{S})} + \frac{1}{\text{poly}(nT)}$.*

Putting things together, we have the guarantees of Theorem 5 for our G-sampler.

3 SGD Algorithm and Analysis

Before introducing our main SGD algorithm, we recall the following algorithm, that essentially outputs noisy version of the rows with high "importance". Although Sampler outputs a (noisy) vector according to the desired probability distribution, we also require an algorithm that automatically does this for indices $i \in [n]$ that are likely to be sampled multiple times across the T iterations. Equivalently, we require explicitly storing the rows with high sensitivities.

▶ **Theorem 16** ([5])**.** *Given a constant $\varepsilon > 0$, there exists an algorithm Sens that returns all indices $i \in [n]$ such that $\sup_x \frac{|\mathbf{a}_i(\langle \mathbf{a}_i, \mathbf{x}\rangle - b_i)|}{\sum_{j=1}^n |\mathbf{a}_j(\langle \mathbf{a}_j, \mathbf{x}\rangle - b_j)|} \geq \frac{1}{200Td}$ for some $\mathbf{x} \in \mathbb{R}^n$, along with the vector $\mathbf{a}_i(\langle \mathbf{a}_i, \mathbf{x}\rangle - b_i)$. The algorithm requires a single pass over $\mathbf{A} = \mathbf{a}_1 \circ \ldots \circ \mathbf{a}_n$, uses $\tilde{\mathcal{O}}\left(nd^2 + Td^2\right)$ runtime and $\tilde{\mathcal{O}}\left(Td^2\right)$ space, and succeeds with probability $1 - \frac{1}{\text{poly}(n)}$.*

The quantity $\sup_x \frac{\|\mathbf{a}_i(\langle \mathbf{a}_i, \mathbf{x}\rangle - b_i)\|_2}{\sum_{j=1}^n \|\mathbf{a}_j(\langle \mathbf{a}_j, \mathbf{x}\rangle - b_j)\|_2}$ can be considered the *sensitivity* of row $\mathbf{a}_i$ and can be interpreted as a measure of "importance" of the row $\mathbf{a}_i$ with respect to the other rows of $\mathbf{A}$.

Algorithm 2 G-sampler for a single class of rows.

Input: Rows $\mathbf{a}_1, \ldots, \mathbf{a}_n$ of a matrix $\mathbf{A} \in \mathbb{R}^{n \times d}$ with $2^k \leq \|a_i\|_2 < 2^{k+1}$ for all $i \in [n]$, function G, accuracy parameter α for sampling parameter ε

Output: Noisy row $\mathbf{v}$ with the correct sampling distribution induced by G

γ uniformly at random from $[1/2, 1]$, $K \leftarrow \mathcal{O}\left(\frac{\log n}{\alpha}\right)$, $L \leftarrow \mathcal{O}(\log n)$

for $\ell \in [L]$ **do** ▷Processing stage

  Form a stream S_ℓ by sampling each row with probability $2^{-\ell+1}$

  Run $\textsc{CountSketch}_\ell^{(1)}$ with threshold $\mathcal{O}\left(\frac{\alpha^3}{\log n}\right)$ and failure probability $\frac{1}{\text{poly}(n,T)}$ by creating a table $A_\ell^{(1)}$ with entries $\mathbf{a}_j^\top \mathbf{a}_j$ in S_ℓ and a table $B_\ell^{(1)}$ with entries $\mathbf{a}_j$ ▷Identify heavy-hitters

  Run $\textsc{CountSketch}_\ell^{(2)}$ with threshold $\mathcal{O}\left(\frac{\alpha^3}{\log n}\right)$ and failure probability $\frac{1}{\text{poly}(n,T)}$ by creating a table $A_\ell^{(2)}$ with entries $\mathbf{a}_j^\top \mathbf{a}_j$ in S_ℓ and a table $B_\ell^{(2)}$ with entries $\mathbf{a}_j$ and separately considering coordinates after post-processing ▷Unbiased estimates of heavy-hitters, see Lemma 11

for $\ell \in [L]$ **do** ▷Post-processing

  Set $C_\ell^{(i)} = A_\ell^{(i)}\mathbf{x} + B_\ell^{(i)}$ with post-multiplication by $\mathbf{x}$ for $i \in \{1, 2\}$

  Query $\widehat{M} \in [M/2, 2M]$, where $M = \sum_{i=1}^n \|G(\langle \mathbf{a}_i, \mathbf{x}\rangle - b_i, \mathbf{a}_i)\|_2$

  for $j \in [K]$ **do**

    if $j > \log_{(1+\alpha)} \frac{\log^2 n}{\alpha^3}$ **then**

      Add $\mathcal{O}\left(\frac{(1+\alpha)^j \alpha^3}{\log n}\right)$ dummy rows that each contribute $\mathcal{O}\left(\frac{\widehat{M}}{(1+\alpha)^j \alpha^2}\right)$ to F_G

  Let $H_\ell^{(i)}$ be the heavy rows of $C_\ell^{(i)}$ for $i \in \{1, 2\}$ from $\textsc{CountSketch}_\ell^{(i)}$

for $j \in [K]$ **do**

  $L_j \leftarrow \max\left(1, \log \frac{\alpha^2 (1+\alpha)^j}{\log n}\right)$

  Let X_j be the estimated heavy-hitters $\mathbf{v}$ from $H_j^{(2)}$ that are reported by $H_j^{(1)}$ with $G(\mathbf{v})$ in $\left[\frac{8\gamma\widehat{M}}{(1+\alpha)^{j+1}}, \frac{8\gamma\widehat{M}}{(1+\alpha)^j}\right)$

  if $L_j = 1$ **then**

    $\widetilde{F_G}(\widetilde{S_j}) \leftarrow \sum_{\mathbf{v} \in X_j} \frac{8\gamma\widehat{M}}{(1+\alpha)^{j+1}}$

  else if $L_j > 1$ and $|X_j| > \frac{1}{\alpha^2}$ **then**

    $\widetilde{F_G}(\widetilde{S_j}) \leftarrow \sum_{\mathbf{v} \in X_j} \frac{8\gamma\widehat{M}}{(1+\alpha)^{j+1}} \cdot 2^{L_j}$

  else

    $\widetilde{F_G}(\widetilde{S_j}) \leftarrow 0$

Sample $j \in [K]$ with probability $\frac{\widetilde{F_G}(\widetilde{S_j})}{\sum \widetilde{F_G}(\widetilde{S_j})}$

Sample $\mathbf{v}$ from X_j with probability $\frac{1}{X_j}$

if $\mathbf{v}$ is a dummy row **then**

  return $\perp$

else

  return $\mathbf{v}$

We now proceed to describe our main SGD algorithm. For the finite-sum optimization problem $\min_{\mathbf{x} \in \mathbb{R}^d} F(\mathbf{x}) := \frac{1}{n}\sum_{i=1}^n G(\langle \mathbf{a}_i, \mathbf{x}\rangle - b_i, \mathbf{a}_i)$, where each G is a piecewise function of a polynomial with degree at most 1, recall that we could simply use an instance of Sampler as an oracle for SGD with importance sampling. However, naïvely running T SGD steps

requires T independent instances, which uses Tnd runtime by Theorem 5. Thus, as our main theoretical contribution, we use a two level data structure by first implicitly partitioning the rows of matrix $\mathbf{A} = \mathbf{a}_1 \circ \ldots \circ \mathbf{a}_n$ into $\beta := \Theta(T)$ buckets $B_1, \ldots, B_\beta$ and creating an instance of ESTIMATOR and SAMPLER for each bucket. The idea is that for a given query $\mathbf{x}_t$ in SGD iteration $t \in [T]$, we first query $\mathbf{x}_t$ to each of the ESTIMATOR data structures to estimate $\sum_{i \in B_j} G(\langle \mathbf{a}_i, \mathbf{x} \rangle - b_i, \mathbf{a}_i)$ for each $j \in [\beta]$. We then sample index $j \in [\beta]$ among the buckets $B_1, \ldots, B_\beta$ with probability roughly $\frac{\sum_{i \in B_j} G(\langle \mathbf{a}_i, \mathbf{x} \rangle - b_i, \mathbf{a}_i)}{\sum_{i=1}^{n} G(\langle \mathbf{a}_i, \mathbf{x} \rangle - b_i, \mathbf{a}_i)}$. Once we have sampled index j, it would seem that querying the instance SAMPLER corresponding to B_j simulates SGD, since SAMPLER now performs importance sampling on the rows in B_j, which gives the correct overall probability distribution for each row $i \in [n]$. Moreover, SAMPLER has runtime proportional to the sparsity of B_j, so the total runtime across the β instances of SAMPLER is $\tilde{\mathcal{O}}(nd)$.

However, an issue arises when the same bucket B_j is sampled multiple times, as we only create a single instance of SAMPLER for each bucket. We avoid this issue by explicitly accounting for the buckets that are likely to be sampled multiple times. Namely, we show that if $\frac{G(\langle \mathbf{a}_i, \mathbf{x}_t \rangle - b_i, \mathbf{a}_i)}{\sum_{j=1}^{n} G(\langle \mathbf{a}_j, \mathbf{x}_t \rangle - b_j, \mathbf{a}_j)} < \mathcal{O}\left(\frac{1}{T}\right)$ for all $t \in [T]$ and $i \in [n]$, then by Bernstein's inequality, the probability that no bucket B_j is sampled at least $2 \log T$ times is at least $1 - \frac{1}{\text{poly}(T)}$. Thus we use SENS to separate all such rows $\mathbf{a}_i$ whose sensitivities violate this property from their respective buckets and explicitly track the SGD steps in which these rows are sampled.

The natural approach would be to create T samplers for each of the rows with sensitivity at least $\Omega\left(\frac{1}{T}\right)$, ensuring that each of these samplers has access to fresh randomness in each of the T SGD steps. However since the sensitivities sum to $\mathcal{O}(d \log n)$, there can be up to $\mathcal{O}(Td \log n)$ rows with sensitivity at least $\Omega\left(\frac{1}{T}\right)$, so creating T samplers for each of these rows could create up to $\Theta(T^2 d \log n)$ samplers, which is prohibitively expensive in T. Instead, we simply keep each row with sensitivity at least $\Omega\left(\frac{1}{T}\right)$ explicitly, while not including them in the bucket. Due to the monotonicity of sensitivities, the sensitivity of each row may only decrease as the stream progresses. In the case that a row had sensitivity at least $\Omega\left(\frac{1}{T}\right)$ at some point, but then no longer exceeds the threshold at some later point, then the row is given as input to the sampler corresponding to the bucket to which the row hashes and then the explicit storage of the row is deleted. This ensures we need only $\tilde{\mathcal{O}}(Td)$ samplers while still avoiding any sampler from being used multiple times across the T SGD steps. We give the algorithm in full in Algorithm 3.

The key property achieved by Algorithm 3 in partitioning the rows and removing the rows that are likely to be sampled multiple times is that each of the SAMPLER instances are queried at most once.

▶ **Lemma 17.** *With probability at least $\frac{98}{100}$, each $t \in [T]$ uses a different instance of SAMPLER$_j$.*

Theorem 4 then follows from Lemma 17 and the sampling distribution guaranteed by each subroutine in Lemma 15. In particular, Lemma 17 crucially guarantees that each step $t \in [T]$ of SGD will receive a vector with fresh independent randomness. Moreover, we have that each (noisy) vector has small variance and is an unbiased estimate of a subgradient sampled from nearly the optimal importance sampling probability distribution.

▶ **Theorem 4.** *Given an input matrix $\mathbf{A} \in \mathbb{R}^{n \times d}$ whose rows arrive sequentially in a data stream along with the corresponding labels of a measurement vector $\mathbf{b} \in \mathbb{R}^d$, and a measure function M whose derivative is a continuous union of piecewise constant or linear functions, there exists an algorithm that performs T steps of SGD with variance within a constant factor of the optimal sampling distribution. The algorithm uses $\tilde{\mathcal{O}}\left(nd^2 + Td^2\right)$ pre-processing time and Td^2 polylog(Tnd) words of space.*

Algorithm 3 Approximate SGD with Importance Sampling.

Input: Matrix $\mathbf{A} = \mathbf{a}_1 \circ \ldots \circ \mathbf{a}_n \in \mathbb{R}^{n \times d}$, parameter T for number of SGD steps.
Output: T gradient directions.

Preprocessing Stage:
$\beta \leftarrow \Theta(T)$ with a sufficiently large constant in the Θ.
Let $h : [n] \to [\beta]$ be a uniformly random hash function.
Let $\mathbf{B}_j$ be the matrix formed by the rows $\mathbf{a}_i$ of $\mathbf{A}$ with $h(i) = j$, for each $j \in [\beta]$.
Create $\Theta(\log(Td))$ instances $\textsc{Estimator}_j$ and $\textsc{Sampler}_j$ for each $\mathbf{B}_j$ with $j \in [\beta]$ with $\varepsilon = \frac{1}{2}$.
Run $\textsc{Sens}$ to find a set L_0 of rows with sensitivity at least $\Omega\left(\frac{1}{T}\right)$.
Gradient Descent Stage:
Randomly pick starting location $\mathbf{x}_0$
for $t = 1$ to T **do**
    Let q_i be the output of $\textsc{Estimator}_j$ on query $\mathbf{x}_{t-1}$ for each $i \in [\beta]$.
    Sample $j \in [\beta]$ with probability $p_j = \frac{q_j}{\sum_{i \in [\beta]} q_i}$.
    if there exists $i \in L_0$ with $h(i) = j$ **then**
        Use $\textsc{Estimator}_j$, L_0, and $\textsc{Sampler}_j$ to sample gradient $\mathbf{w}_t = \widehat{\nabla f_{i_t}(\mathbf{x}_t)}$
    else
        Use fresh $\textsc{Sampler}_j$ to sample gradient $\mathbf{w}_t = \widehat{\nabla f_{i_t}(\mathbf{x}_t)}$
    $\widehat{p_{i,t}} \leftarrow \frac{\|\mathbf{w}_t\|_2^2}{\sum_{j \in [\beta]} q_j}$
    $\mathbf{x}_{t+1} \leftarrow \mathbf{x}_t - \frac{\eta_t}{n\widehat{p_{i,t}}} \cdot \mathbf{w}_t$

Proof. Consider Algorithm 3. By Lemma 17, each time $t \in [T]$ uses a fresh instance of $\textsc{Sampler}_j$, so that independent randomness is used. A possible concern is that each instance $\textsc{Estimator}_j$ is not using fresh randomness, but we observe that the $\textsc{Estimator}$ procedures are only used in sampling a bucket $j \in [\beta]$; otherwise the sampling uses fresh randomness whereas the sampling is built into each instance of $\textsc{Sampler}_j$. By Theorem 5, each index i is sampled with probability within a factor 2 of the importance sampling probability distribution. By Theorem 9, we have that $\widehat{p_{i,t}}$ is within a factor 4 of the probability $p_{i,t}$ induced by optimal importance sampling SGD. Note that $\mathbf{w}_t = G(\langle \mathbf{a}_i, \widehat{\mathbf{x}_t} \rangle - b_i, \mathbf{a}_i)$ is an unbiased estimator of $G(\langle \mathbf{a}_i, \mathbf{x}_t \rangle - b_i, \mathbf{a}_i)$ and $G(\mathbf{w}_t)$ is a 2-approximation to $G(\mathbf{x}_t)$ by Theorem 5. Hence, the variance at each time $t \in [T]$ of Algorithm 3 is within a constant factor of the variance $\sigma^2 = \left(\sum_{i=1}^n G(\langle \mathbf{a}_i, \mathbf{x}_t \rangle - b_i, \mathbf{a}_i)\right)^2 - \sum_{i=1}^n G(\langle \mathbf{a}_i, \mathbf{x}_t \rangle - b_i, \mathbf{a}_i)^2$ of optimal importance sampling SGD.

By Theorem 5, Theorem 9, and Theorem 16, the preprocessing time is d^2 polylog(nT) for $\varepsilon = \mathcal{O}(1)$ and $\beta = \Theta(T)$, but partitioning the non-zero entries of $\mathbf{A}$ across the β buckets and the space used by the algorithm is $\tilde{\mathcal{O}}\left(Td^2\right)$. Once the gradient descent stage of Algorithm 3 begins, it takes Td^2 polylog(n) time in each step $t \in [T]$ to query the $\beta = \Theta(T)$ instances of $\textsc{Sampler}$ and $\textsc{Estimator}$, for total time Td^2 polylog(n). ◀

Finally, we derandomize our algorithm in Appendix B with an extra logarithmic factor in the space complexity by using the following formulation of Nisan's pseudorandom generator:

▶ **Theorem 18** (Nisan's Pseudorandom Generator). *[28] Let $\mathcal{A}$ be an algorithm that uses $S = \Omega(\log n)$ space and R random bits. Then there exists a pseudorandom generator for $\mathcal{A}$ that succeeds with high probability and runs in $\mathcal{O}(S \log R)$ bits.*

References

1. Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. *J. Comput. Syst. Sci.*, 58(1):137–147, 1999.
2. Alexandr Andoni, Robert Krauthgamer, and Krzysztof Onak. Streaming algorithms via precision sampling. In *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS*, pages 363–372, 2011.
3. Jaroslaw Blasiok, Vladimir Braverman, Stephen R. Chestnut, Robert Krauthgamer, and Lin F. Yang. Streaming symmetric norms via measure concentration. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC*, pages 716–729, 2017.
4. Michael Bosse, Gabriel Agamennoni, and Igor Gilitschenski. Robust estimation and applications in robotics. *Found. Trends Robotics*, 4(4):225–269, 2016.
5. Vladimir Braverman, Petros Drineas, Cameron Musco, Christopher Musco, Jalaj Upadhyay, David P. Woodruff, and Samson Zhou. Near optimal linear algebra in the online and sliding window models. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS*, pages 517–528, 2020.
6. Sébastien Bubeck. Convex optimization: Algorithms and complexity. *Foundations and Trends in Machine Learning*, 8(3-4):231–357, 2015.
7. Moses Charikar, Kevin C. Chen, and Martin Farach-Colton. Finding frequent items in data streams. *Theor. Comput. Sci.*, 312(1):3–15, 2004.
8. Kenneth L. Clarkson, Ruosong Wang, and David P. Woodruff. Dimensionality reduction for tukey regression. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning, ICML*, pages 1262–1271, 2019.
9. Kenneth L. Clarkson and David P. Woodruff. Sketching for M-estimators: A unified approach to robust regression. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 921–939, 2015.
10. Hadi Daneshmand, Aurélien Lucchi, and Thomas Hofmann. Starting small - learning with adaptive sample sizes. In *Proceedings of the 33nd International Conference on Machine Learning, ICML*, pages 1463–1471, 2016.
11. Aaron Defazio, Francis R. Bach, and Simon Lacoste-Julien. SAGA: A fast incremental gradient method with support for non-strongly convex composite objectives. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems*, pages 1646–1654, 2014.
12. Antoine Guitton and William W Symes. Robust and stable velocity analysis using the huber function. In *SEG Technical Program Expanded Abstracts 1999*, pages 1166–1169. Society of Exploration Geophysicists, 1999.
13. Peter J Huber. Robust estimation of a location parameter. In *Breakthroughs in statistics*, pages 492–518. Springer, 1992.
14. Piotr Indyk and David P. Woodruff. Optimal approximations of the frequency moments of data streams. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing*, pages 202–208, 2005.
15. Rajesh Jayaram and David P. Woodruff. Perfect lp sampling in a data stream. In *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS*, pages 544–555, 2018.
16. Rajesh Jayaram, David P. Woodruff, and Samson Zhou. Truly perfect samplers for data streams and sliding windows. In *PODS: International Conference on Management of Data*, pages 29–40, 2022.
17. Yifei Jiang, Yi Li, Yiming Sun, Jiaxin Wang, and David P. Woodruff. Single pass entrywise-transformed low rank approximation. In *Proceedings of the 38th International Conference on Machine Learning, ICML*, pages 4982–4991, 2021.
18. Rie Johnson and Tong Zhang. Accelerating stochastic gradient descent using predictive variance reduction. In *Advances in Neural Information Processing Systems 26, Proceedings.*, pages 315–323, 2013.

19 Tyler B. Johnson and Carlos Guestrin. Training deep models faster with robust, approximate importance sampling. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems, NeurIPS*, pages 7276–7286, 2018.

20 Hossein Jowhari, Mert Saglam, and Gábor Tardos. Tight bounds for lp samplers, finding duplicates in streams, and related problems. In *Proceedings of the 30th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS*, pages 49–58, 2011.

21 Angelos Katharopoulos and François Fleuret. Not all samples are created equal: Deep learning with importance sampling. In *Proceedings of the 35th International Conference on Machine Learning, ICML*, pages 2530–2539, 2018.

22 Sepideh Mahabadi, Ilya P. Razenshteyn, David P. Woodruff, and Samson Zhou. Non-adaptive adaptive sampling on turnstile streams. In *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC*, pages 1251–1264, 2020.

23 Raghu Meka. Cs289ml: Algorithmic machine learning notes, 2017. URL: `https://raghumeka.github.io/CS289ML/gdnotes.pdf`.

24 Morteza Monemizadeh and David P. Woodruff. 1-pass relative-error lp-sampling with applications. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 1143–1160, 2010.

25 Deanna Needell, Nathan Srebro, and Rachel Ward. Stochastic gradient descent, weighted sampling, and the randomized kaczmarz algorithm. *Math. Program.*, 155(1-2):549–573, 2016.

26 Arkadi Nemirovski, Anatoli B. Juditsky, Guanghui Lan, and Alexander Shapiro. Robust stochastic approximation approach to stochastic programming. *SIAM Journal on Optimization*, 19(4):1574–1609, 2009.

27 Arkadi Semenovich Nemirovsky and David Borisovich Yudin. Problem complexity and method efficiency in optimization, 1983.

28 Noam Nisan. Pseudorandom generators for space-bounded computation. *Combinatorica*, 12(4):449–461, 1992.

29 Eric Price, Sandeep Silwal, and Samson Zhou. Hardness and algorithms for robust and sparse optimization. In *International Conference on Machine Learning, ICML*, pages 17926–17944, 2022.

30 Xun Qian, Peter Richtárik, Robert M. Gower, Alibek Sailanbayev, Nicolas Loizou, and Egor Shulgin. SGD with arbitrary sampling: General analysis and improved rates. In *Proceedings of the 36th International Conference on Machine Learning, ICML*, volume 97, pages 5200–5209, 2019.

31 Sashank J. Reddi, Ahmed Hefny, Suvrit Sra, Barnabás Póczos, and Alexander J. Smola. On variance reduction in stochastic gradient descent and its asynchronous variants. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems*, pages 2647–2655, 2015.

32 Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

33 Nicolas Le Roux, Mark Schmidt, and Francis R. Bach. A stochastic gradient method with an exponential convergence rate for finite training sets. In *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems.*, pages 2672–2680, 2012.

34 Farnood Salehi, Patrick Thiran, and L. Elisa Celis. Coordinate descent with bandit sampling. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems, NeurIPS*, pages 9267–9277, 2018.

35 Sebastian U. Stich, Anant Raj, and Martin Jaggi. Safe adaptive importance sampling. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems*, pages 4381–4391, 2017.

36 Mikkel Thorup and Yin Zhang. Tabulation based 4-universal hashing with applications to second moment estimation. In *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 615–624. SIAM, 2004.

37 Murad Tukan, Xuan Wu, Samson Zhou, Vladimir Braverman, and Dan Feldman. New coresets for projective clustering and applications. In *International Conference on Artificial Intelligence and Statistics, AISTATS*, 2022.

38 Ming Yuan and Yi Lin. Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 68(1):49–67, 2006.

39 Zhengyou Zhang. Parameter estimation techniques: a tutorial with application to conic fitting. *Image Vis. Comput.*, 15(1):59–76, 1997.

40 Peilin Zhao and Tong Zhang. Stochastic optimization with importance sampling for regularized loss minimization. In *Proceedings of the 32nd International Conference on Machine Learning ICML*, pages 1–9, 2015.

A Missing Proofs from Section 2

Proof of Lemma 11. Given $\mathbf{A} \in \mathbb{R}^{n\times d}$, let $\mathbf{A}_i = \mathbf{a}_i^\top \mathbf{a}_i$ for all $i \in [n]$. For a fixed coordinate $k \in [d]$, we define a vector $\mathbf{v}^{(k)} \in \mathbb{R}^n$ so that for each $i \in [n]$, the i-th coordinate of $\mathbf{v}^{(k)}$ is the k-th coordinate of $\mathbf{A}_i\mathbf{x} \in \mathbb{R}^d$.

Suppose we run a separate CountSketch instance on $\mathbf{v}^{(k)}$. For a fixed index $i \in [n]$, let $h(i)$ be the bucket of $\mathcal{T}$ to which $v_i^{(k)}$ hashes. For each $j \in [n]$, let I_j be the indicator variable for whether $v_j^{(k)}$ also hashes to bucket $h(i)$, so that $I_j = 1$ if $h(i) = h(j)$ and $I_j = 0$ if $h(i) \neq h(j)$. Similarly for each $j \in [n]$, let s_j be a random sign assigned to j, so that the estimate for $v_i^{(k)}$ by a single row of CountSketch is

$$\sum_{j\in[n]} s_i s_j I_j v_j^{(k)} = v_i^{(k)} + \sum_{j:h(j)=h(i)} r_j v_j^{(k)},$$

where $r_j = s_i s_j$ satisfies $r_j = 1$ with probability $\frac{1}{2}$ and $r_j = -1$ with probability $\frac{1}{2}$. Thus if y_i is the estimate for $v_i^{(k)}$, then for any real number u, we have that

$$\mathbf{Pr}\left[y_i = v_i^{(k)} + u\right] = \mathbf{Pr}\left[y_i = v_i^{(k)} - u\right],$$

so that the probability mass function of y_i is symmetric about $v_i^{(k)}$. Thus given ℓ independent instances of CountSketch with estimates $y_i^{(k,1)}, \ldots, y_i^{(k,\ell)}$ for $v_i^{(k)}$ and any real numbers $u^{(1)}, \ldots, u^{(\ell)}$,

$$\mathbf{Pr}\left[y_i^{(k,1)} = v_i^{(k)} + u^{(1)}, \ldots, y_i^{(k,\ell)} = v_i^{(k)} + u^{(\ell)}\right] = \mathbf{Pr}\left[y_i^{(k,1)} = v_i^{(k)} - u^{(1)}, \ldots, y_i^{(k,\ell)} = v_i^{(k)} - u^{(\ell)}\right].$$

Therefore, the joint probability mass function is symmetric about $(v_i^{(k)}, \ldots, v_i^{(k)})$ and so the median across the ℓ instances of CountSketch is an unbiased estimator to $v_i^{(k)}$. Finally, we have due to the properties of CountSketch that if each hash function h maps to a universe of size $\mathcal{O}\left(\frac{1}{\varepsilon^2}\right)$ and $\ell = \mathcal{O}\left(\log(nT)\right)$, then with probability at least $1 - \frac{1}{\mathrm{poly}(T,n)}$, the output estimate for $v_i^{(k)}$ has additive error at most $\varepsilon \cdot \left(\sum_{j\in\mathrm{tail}(2/\varepsilon^2)}(v_i^{(k)})^2\right)^{1/2}$.

Thus using each of the estimated outputs across all $k \in [d]$, then for a fixed $i \in [n]$, we can output a vector $\mathbf{y}_i$ such that $\mathbb{E}\left[\mathbf{y}_i\right] = \mathbf{A}_i\mathbf{x}$ and with probability at least $1 - \frac{1}{\mathrm{poly}(T,n)}$,

$$\left|\, \|\mathbf{y}_i\|_2 - \|\mathbf{A}_i\mathbf{x}\|_2 \,\right| \leq \varepsilon \cdot \left(\sum_{i\in\mathrm{tail}(2/\varepsilon^2)} \|\mathbf{A}_i\mathbf{x}\|_2^2\right)^{1/2}.$$

For a fixed $k \in [d]$, then our algorithm intends to hash the k-th coordinate of $\mathbf{A}_i\mathbf{x} \in \mathbb{R}^d$. However, since $\mathbf{x}$ is only given after the data structure is already formed and in particular, after $\mathbf{A}_i$ is given, then CountSketch must hash the k-th row of $\mathbf{A}_i$ entirely, thus storing $\mathcal{O}\left(\frac{d}{\varepsilon^2}\log^2(nT)\right)$ bits for each coordinate $k \in [d]$. Hence across all $k \in [d]$, the algorithm uses the total space $\mathcal{O}\left(\frac{d^2}{\varepsilon^2}\log^2(nT)\right)$. ◀

Proof of Lemma 12. Since $\widetilde{S}$ includes all the rows of S, then $F_G(\widetilde{S}) \geq F_G(S)$. Since each level $j \in [K]$ acquires $\mathcal{O}\left(\frac{(1+\alpha)^j\alpha^3}{\log n}\right)$ dummy rows that each contribute $\mathcal{O}\left(\frac{\widehat{M}}{(1+\alpha)^j\alpha^2}\right)$ to F_G in $\widetilde{S}$, then each level of $F_G(S)$ contributes at most $\mathcal{O}\left(\frac{\widehat{M}\cdot\alpha}{\log n}\right)$ more to $F_G(\widetilde{S})$. Because $K = \mathcal{O}\left(\frac{\log n}{\alpha}\right)$, then the total additional contribution by the dummy rows is at most $\mathcal{O}\left(\widehat{M}\right)$. Since $\widehat{M} \leq 2F_G(S)$, then it follows that for sufficiently small constant in the contribution of each dummy row, we have $F_G(\widetilde{S}) - F_G(S) \leq F_G(S)$ and thus, $F_G(\widetilde{S}) \leq 2F_G(S)$. ◀

Proof of Lemma 13. Observe that the number of rows that exceed $\frac{\widehat{M}}{2^j}$ is at most 2^{j+1}. Thus the expected number of rows that exceed $\frac{\widehat{M}}{2^j}$ sampled by S_j is at most $\frac{1}{2}$. Hence by Chernoff bounds, the probability that the number of rows that exceed $\frac{\widehat{M}}{2^j}$ sampled by S_j is more than $t = \mathcal{O}\left(\frac{\log n}{\alpha^3}\right)$ is $\frac{1}{\text{poly}(nT)}$. ◀

Proof of Lemma 14. Suppose that for each $j \in [K]$, level j consists of N_j rows and note that $N_j \geq \mathcal{O}\left(\frac{(1+\alpha)^j\alpha^3}{\log n}\right)$ elements due to the dummy rows. Each element is sampled with some probability p_{L_j}, where $L_j = \max\left(1, \log\frac{\alpha^2(1+\alpha)^j}{\log n}\right)$ and thus $p_{L_j}(1+\alpha)^j > 1$ since $p_{L_j} = \frac{1}{2^{L_j}}$. Let $\widehat{N_j}$ be the number of items sampled in $\widetilde{S_{L_j}}$. We have $\mathbb{E}\left[2^{L_j}\cdot\widehat{N_j}\right] = N_j$ and the second moment is at most $N_j \cdot 2^{L_j} \leq \frac{\alpha^2}{\log n}(N_j)^2$. Thus by Chernoff bounds with $\mathcal{O}(\log n)$-wise independence, we have that with high probability,

$$(1 - \mathcal{O}(\alpha))N_j \leq 2^{L_j}\cdot\widehat{N_j} \leq (1 + \mathcal{O}(\alpha))N_j.$$

Each estimated row norm is a $(1+\alpha)$-approximation to the actual row norm due to Lemma 13. Thus by Lemma 12, we have that $F_G(\widetilde{S_j}) \leq 2F_G(S_j)$ so that each of the $\widehat{N_j}$ rows will be detected by the threshold of CountSketch with the tail guarantee, i.e., Lemma 11. Moreover, we assume that a noisy row with $(1+\alpha)$-approximation to the row norm of the original vector suffices to obtain a $(1+\varepsilon)$-approximation to the contribution of the row. Therefore, the result then follows in an ideal scenario where $G(\mathbf{v}) \in \left[\frac{\widehat{M}}{2^j}, \frac{2\widehat{M}}{2^j}\right)$ if and only if the corresponding row $\mathbf{a}_i$ satisfies $G(\mathbf{a}_i) \in \left[\frac{\widehat{M}}{2^j}, \frac{2\widehat{M}}{2^j}\right)$. Unfortunately, this may not be true because $G(\mathbf{a}_i)$ may lie near the boundary of the interval $\left[\frac{\widehat{M}}{2^j}, \frac{2\widehat{M}}{2^j}\right)$ while the estimate $G(\mathbf{v})$ has a value that does not lie within the interval. In this case, $G(\mathbf{v})$ is used toward the estimation of some other level set.

Hence, our algorithm randomizes the boundaries of the level sets $\left[\frac{4\gamma\widehat{M}}{2^j}, \frac{8\gamma\widehat{M}}{2^j}\right)$ by choosing $\gamma \in [1/2, 1)$ uniformly at random. Since the threshold of CountSketch is $\mathcal{O}\left(\frac{\alpha^3}{\log n}\right)$ then the probability that each row $\mathbf{a}_i$ is misclassified over the choice of γ is at most $\mathcal{O}(\varepsilon)$. Moreover, if $\mathbf{a}_i$ is misclassified, then its contribution can only be classified into level set $j-1$ or $j+1$, inducing an incorrect multiplicative factor of at most two. Hence, the error due to the misclassification across all rows is at most $\mathcal{O}(\varepsilon)$ fraction of $F_G(S_j)$ in expectation. By Markov's inequality, this error is a most ε-fraction of $F_G(S_j)$ with probability at least $3/4$. Then by taking the median across $\mathcal{O}(\log(nT))$ independent instances, we obtain high probability of success. ◀

Proof of Lemma 15. Conditioned on the correctness of each of the estimates $\widetilde{F_G}(\widetilde{S_j})$, which occurs with high probability by Lemma 14, the probability that the algorithm selects $j \in [K]$ is $\frac{\widetilde{F_G}(\widetilde{S_j})}{\sum_{j\in[K]}\widetilde{F_G}(\widetilde{S_j})}$. Conditioned on the algorithm selecting $j \in [K]$, then either the algorithm will choose a dummy row, or it will choose a row uniformly at random from the rows $\mathbf{v} \in X_j$, where X_j is the set of heavy-hitters reported by H_j with L_2 norm in $\left[\frac{8\gamma\widehat{M}}{(1+\alpha)^{j+1}}, \frac{8\gamma\widehat{M}}{(1+\alpha)^j}\right)$. The latter event occurs with probability $\frac{\widetilde{F_G}(S_j)}{\widetilde{F_G}(\widetilde{S_j})}$. Due to the tail guarantee of CountSketch in Lemma 11, we have that each heavy hitter $\mathbf{v} \in X_j$ corresponds to a vector $\mathbf{a}_i(\langle \mathbf{a}_i, x\rangle - b_i)$ such that $\|\mathbf{v} - \mathbf{a}_i(\langle \mathbf{a}_i, x\rangle - b_i)\|_2 \le \varepsilon\|\mathbf{a}_i(\langle \mathbf{a}_i, x\rangle - b_i)\|\|_2$. Moreover, by Lemma 11, we have that $\mathbb{E}[\mathbf{v}] = \mathbf{a}_i(\langle \mathbf{a}_i, x\rangle - b_i)$. Hence the probability that vector $\mathbf{v}$ is selected is $\frac{(1\pm\mathcal{O}(\alpha))G(\mathbf{a}_i(\langle \mathbf{a}_i, x\rangle - b_i)}{\widetilde{F_G}(S_j)}$. ◀

Putting things together, we have the guarantees of Theorem 5 for our G-sampler.

▶ **Theorem 5.** *Given an (α, ε)-smooth gradient G, there exists an algorithm Sampler that outputs a noisy vector $\mathbf{v}$ such that $\|\mathbf{v} - \mathbf{a}_i(\langle \mathbf{a}_i, x\rangle - b_i)\|_2 \le \alpha\|\mathbf{a}_i(\langle \mathbf{a}_i, x\rangle - b_i)\|\|_2$ and $\mathbb{E}[\mathbf{v}] = \mathbf{a}_i(\langle \mathbf{a}_i, x\rangle - b_i)$ is $(1 \pm \mathcal{O}(\varepsilon))\frac{\|G(\langle \mathbf{a}_i, \mathbf{x}\rangle - b_i, \mathbf{a}_i)\|_2}{\sum_{j\in[n]}\|G(\langle \mathbf{a}_j, \mathbf{x}\rangle - b_j, \mathbf{a}_j)\|_2} + \frac{1}{\mathrm{poly}(n)}$. The algorithm uses d^2 poly$\left(\log(nT), \frac{1}{\alpha}\right)$ update time per arriving row and Td^2 poly$\left(\log(nT), \frac{1}{\alpha}\right)$ total bits of space.*

Proof. We define a class C_k of rows as the subset of rows of the input matrix $\mathbf{A}$ such that $2^k \le \|\mathbf{a}_i\|_2 < 2^{k+1}$. We first use the estimator algorithm in Theorem 9 to sample a class k of rows with probability $\frac{\sum_{\mathbf{a}_i\in C_k} G(\langle \mathbf{a}_i, x\rangle - b_i, \mathbf{a}_i)}{\sum_{j\in[n]} G(\langle \mathbf{a}_j, x\rangle - b_j, \mathbf{a}_j)}$. Once a class C_k is selected, then outputting a row from C_k under the correct distribution follows from Lemma 15. The space complexity follows from storing a $d \times d$ matrix in each of the $\mathcal{O}\left(\frac{\log^2(nT)}{\alpha^3}\right)$ buckets in CountSketch for threshold $\mathcal{O}\left(\frac{\alpha^3}{\log(nT)}\right)$ and high probability of success. ◀

B Missing Proofs from Section 3

Proof of Lemma 17. Let $C > 0$ be a sufficiently large constant. For any $t \in [T]$ and $i \in [n]$, $G(\mathbf{a}_i(\langle \mathbf{a}_i, x\rangle - b_i)) \ge \frac{1}{CT}\sum_{j\in[n]} G(\mathbf{a}_j(\langle \mathbf{a}_j, x\rangle - b_j))$ only if there exists a row in $\mathbf{a}_i \circ b_i$ whose sensitivity is at least $\frac{1}{CT}$. However, we have explicitly stored all rows $\mathbf{a}_i \circ b_i$ with sensitivity $\Omega\left(\frac{1}{T}\right)$ and removed them from each G-sampler.

Thus, for all $j \in [\beta]$ so that $h(i) \ne j$ for any index $i \in [n]$ such that $G(\mathbf{a}_i(\langle \mathbf{a}_i, x\rangle - b_i)) \le \frac{1}{CT}\sum_{k\in[n]} G(\mathbf{a}_k(\langle \mathbf{a}_k, x\rangle - b_k))$, we have

$$\sum_{i:h(i)=j} G(\mathbf{a}_i(\langle \mathbf{a}_i, x\rangle - b_i)) \le \frac{\log(Td)}{200T}\sum_{k\in[n]} G(\mathbf{a}_k(\langle \mathbf{a}_k, x\rangle - b_k)),$$

with probability at least $1 - \frac{1}{\mathrm{poly}(Td)}$ by Bernstein's inequality and a union bound over $j \in [\beta]$, where $\beta = \Theta(T)$ is sufficiently large. Intuitively, by excluding the hash indices containing "heavy" matrices, the remaining hash indices contain only a small fraction of the mass with high probability.

We analyze the probability that any bucket containing rows with sensitivity less than $\mathcal{O}\left(\frac{1}{T}\right)$ are sampled more than $\Omega(T\log(Td))$ times, since we create $\mathcal{O}(T\log(Td))$ separate G-samplers for each of these buckets. By a coupling argument and Chernoff bounds, the probability that any $j \in [\beta]$ with $\sum_{i:h(i)=j} G(\mathbf{a}_i(\langle \mathbf{a}_i, x\rangle - b_i)) \le \frac{\log(Td)}{200T}\sum_{k\in[n]} G(\mathbf{a}_k(\langle \mathbf{a}_k, x\rangle -$

b_k)) is sampled more than $200\log(Td)$ times is at most $\frac{1}{\text{poly}(Td)}$ for any $t \in [T]$, provided there is no row with $h(i) = j$ whose sensitivity is at least $\frac{1}{CT}$. Thus, the probability that some bucket $j \in [\beta]$ is sampled more than $200\log(Td)$ times across T steps is at most $\frac{1}{\text{poly}(Td)}$.

In summary, we would like to maintain T separate instances of G-samplers for the heavy matrices and $\Theta(\log(Td))$ separate instances of G-samplers for each hash index that does not contain a heavy matrix, but this creates a $\Omega(T^2)$ space dependency. Instead, we explicitly store the heavy rows with sensitivity $\Omega\left(\frac{1}{T}\right)$, removing them from the heavy matrices, and manually perform the sampling, rather than rely on the G-sampler subroutine. There can be at most $\mathcal{O}(Td\log n)$ such rows, resulting in $\mathcal{O}\left(Td^2\log n\right)$ overall space for storing these rows explicitly. Since the resulting matrices are light by definition, we can maintain $\Theta(\log(Td))$ separate instances of G-samplers for each of the $\Theta(T)$ buckets, which results in $\tilde{\mathcal{O}}\left(Td^2\right)$ space overall. With probability at least $\frac{98}{100}$, any hash index not containing a heavy matrix is sampled only once, so each time $t \in [T]$ has access to a fresh G-sampler. ◀

Derandomization of the algorithm. To derandomize our algorithm, we first recall the following formulation of Nisan's pseudorandom generator.

▶ **Theorem 19** (Nisan's Pseudorandom Generator, [28]). *Let $\mathcal{A}$ be an algorithm that uses $S = \Omega(\log n)$ space and R random bits. Then there exists a pseudorandom generator for $\mathcal{A}$ that succeeds with high probability and runs in $\mathcal{O}(S\log R)$ bits.*

The goal of Nisan's PRG is to fool a small space tester by generating a number of pseudorandom bits in a read-once tape in place of a number of truly random bits. In the row-arrival model, the updates to each row $\mathbf{a}_i$ of $\mathbf{A} \in \mathbb{R}^{n\times d}$ arrive sequentially, so it suffices to use a read-once input tape. Thus a tester that is only allowed to S space cannot distinguish between the output of our algorithm using true randomness and pseudorandom bits generated by Nisan's PRG. Since our algorithm uses $S = Td^2\,\text{polylog}(Tnd)$ bits of space and $R = \text{poly}(n, T, d)$ bits of randomness, then it can be randomized by Nisan's PRG while using $Td^2\,\text{polylog}(Tnd)$ total space.