FetchSGD: Communication-Efficient Federated Learning with Sketching

Daniel Rothchild[*1] **Ashwinee Panda**[*1] **Enayat Ullah**[2] **Nikita Ivkin**[3] **Ion Stoica**[1] **Vladimir Braverman**[2] **Joseph Gonzalez**[1] **Raman Arora**[2]

Abstract

Existing approaches to federated learning suffer from a communication bottleneck as well as convergence issues due to sparse client participation. In this paper we introduce a novel algorithm, called `FetchSGD`, to overcome these challenges. `FetchSGD` compresses model updates using a Count Sketch, and then takes advantage of the mergeability of sketches to combine model updates from many workers. A key insight in the design of `FetchSGD` is that, because the Count Sketch is linear, momentum and error accumulation can both be carried out within the sketch. This allows the algorithm to move momentum and error accumulation from clients to the central aggregator, overcoming the challenges of sparse client participation while still achieving high compression rates and good convergence. We prove that `FetchSGD` has favorable convergence guarantees, and we demonstrate its empirical effectiveness by training two residual networks and a transformer model.

1. Introduction

Federated learning has recently emerged as an important setting for training machine learning models. In the federated setting, training data is distributed across a large number of edge devices, such as consumer smartphones, personal computers, or smart home devices. These devices have data that is useful for training a variety of models – for text prediction, speech modeling, facial recognition, document identification, and other tasks (Shi et al., 2016; Brisimi et al., 2018; Leroy et al., 2019; Tomlinson et al., 2009). However, data privacy, liability, or regulatory concerns may make it difficult to move this data to the cloud for training (EU, 2018). Even without these concerns, training machine learning models in the cloud can be expensive, and an effective way to train the same models on the edge has the potential to eliminate this expense.

When training machine learning models in the federated setting, participating clients do not send their local data to a central server; instead, a central aggregator coordinates an optimization procedure among the clients. At each iteration of this procedure, clients compute gradient-based updates to the current model using their local data, and they communicate only these updates to a central aggregator.

A number of challenges arise when training models in the federated setting. Active areas of research in federated learning include solving systems challenges, such as handling stragglers and unreliable network connections (Bonawitz et al., 2016; Wang et al., 2019), tolerating adversaries (Bagdasaryan et al., 2018; Bhagoji et al., 2018), and ensuring privacy of user data (Geyer et al., 2017; Hardy et al., 2017). In this work we address a different challenge, namely that of training high-quality models under the constraints imposed by the federated setting.

There are three main constraints unique to the federated setting that make training high-quality models difficult. First, **communication-efficiency** is a necessity when training on the edge (Li et al., 2018), since clients typically connect to the central aggregator over slow connections ($\sim$ 1Mbps) (Lee et al., 2010). Second, **clients must be stateless,** since it is often the case that no client participates more than once during all of training (Kairouz et al., 2019). Third, the **data collected across clients is typically not independent and identically distributed.** For example, when training a next-word prediction model on the typing data of smartphone users, clients located in geographically distinct regions generate data from different distributions, but enough commonality exists between the distributions that we may still want to train a single model (Hard et al., 2018; Yang et al., 2018).

In this paper, we propose a new optimization algorithm for federated learning, called `FetchSGD`, that can train high-quality models under all three of these constraints. The crux of the algorithm is simple: at each round, clients compute a gradient based on their local data, then compress the gradient using a data structure called a Count Sketch before

[*]Equal contribution [1]University of California, Berkeley, California, USA [2]Johns Hopkins University, Baltimore, Maryland [3]Amazon. Correspondence to: Daniel Rothchild <drothchild@berkeley.edu>.

Proceedings of the 37[th] International Conference on Machine Learning, Vienna, Austria, PMLR 119, 2020.

sending it to the central aggregator. The aggregator maintains momentum and error accumulation Count Sketches, and the weight update applied at each round is extracted from the error accumulation sketch. See Figure 1 for an overview of FetchSGD.

FetchSGD requires no local state on the clients, and we prove that it is communication efficient, and that it converges in the non-i.i.d. setting for L-smooth non-convex functions at rates $\mathcal{O}\left(T^{-1/2}\right)$ and $\mathcal{O}\left(T^{-1/3}\right)$ respectively under two alternative assumptions – the first opaque and the second more intuitive. Furthermore, even without maintaining any local state, FetchSGD can carry out momentum – a technique that is essential for attaining high accuracy in the non-federated setting – as if on local gradients before compression (Sutskever et al., 2013). Lastly, due to properties of the Count Sketch, FetchSGD scales seamlessly to small local datasets, an important regime for federated learning, since user interaction with online services tends to follow a power law distribution, meaning that most users will have relatively little data to contribute (Muchnik et al., 2013).

We empirically validate our method with two image recognition tasks and one language modeling task. Using models with between 6 and 125 million parameters, we train on non-i.i.d. datasets that range in size from 50,000 – 800,000 examples.

2. Related Work

Broadly speaking, there are two optimization strategies that have been proposed to address the constraints of federated learning: Federated Averaging (FedAvg) and extensions thereof, and gradient compression methods. We explore these two strategies in detail in Sections 2.1 and 2.2, but as a brief summary, FedAvg does not require local state, but it also does not reduce communication from the standpoint of a client that participates once, and it struggles with non-i.i.d. data and small local datasets because it takes many local gradient steps. Gradient compression methods, on the other hand, can achieve high communication efficiency. However, it has been shown both theoretically and empirically that these methods must maintain error accumulation vectors on the clients in order to achieve high accuracy. This is ineffective in federated learning, since clients typically participate in optimization only once, so the accumulated error has no chance to be re-introduced (Karimireddy et al., 2019b).

2.1. FedAvg

FedAvg reduces the total number of bytes transferred during training by carrying out multiple steps of stochastic gradient descent (SGD) locally before sending the aggregate model update back to the aggregator. This technique, often referred to as local/parallel SGD, has been studied since the early days of distributed model training in the data center (Dean et al., 2012), and is referred to as FedAvg when applied to federated learning (McMahan et al., 2016). FedAvg has been successfully deployed in a number of domains (Hard et al., 2018; Li et al., 2019), and is the most commonly used optimization algorithm in the federated setting (Yang et al., 2018). In FedAvg, every participating client first downloads and trains the global model on their local dataset for a number of epochs using SGD. The clients upload the difference between their initial and final model to the parameter server, which averages the local updates weighted according to the magnitude of the corresponding local dataset.

One major advantage of FedAvg is that it requires no local state, which is necessary for the common case where clients participate only once in training. FedAvg is also communication-efficient in that it can reduce the total number of bytes transferred during training while achieving the same overall performance. However, from an individual client's perspective, there is no communication savings if the client participates in training only once. Achieving high accuracy on a task often requires using a large model, but clients' network connections may be too slow or unreliable to transmit such a large amount of data at once (Yang et al., 2010).

Another disadvantage of FedAvg is that taking many local steps can lead to degraded convergence on non-i.i.d. data. Intuitively, taking many local steps of gradient descent on local data that is not representative of the overall data distribution will lead to local over-fitting, which will hinder convergence (Karimireddy et al., 2019a). When training a model on non-i.i.d. local datasets, the goal is to minimize the average test error across clients. If clients are chosen randomly, SGD naturally has convergence guarantees on non-i.i.d. data, since the average test error is an expectation over which clients participate. However, although FedAvg has convergence guarantees for the i.i.d. setting (Wang and Joshi, 2018), these guarantees do not apply directly to the non-i.i.d. setting as they do with SGD. Zhao et al. (2018) show that FedAvg, using K local steps, converges as $\mathcal{O}\left(K/T\right)$ on non-i.i.d. data for strongly convex smooth functions, with additional assumptions. In other words, convergence on non-i.i.d. data could slow down as much as proportionally to the number of local steps taken.

Variants of FedAvg have been proposed to improve its performance on non-i.i.d. data. Sahu et al. (2018) propose constraining the local gradient update steps in FedAvg by penalizing the L2 distance between local models and the current global model. Under the assumption that every client's loss is minimized wherever the overall loss function is minimized, they recover the convergence rate of SGD. Karim-

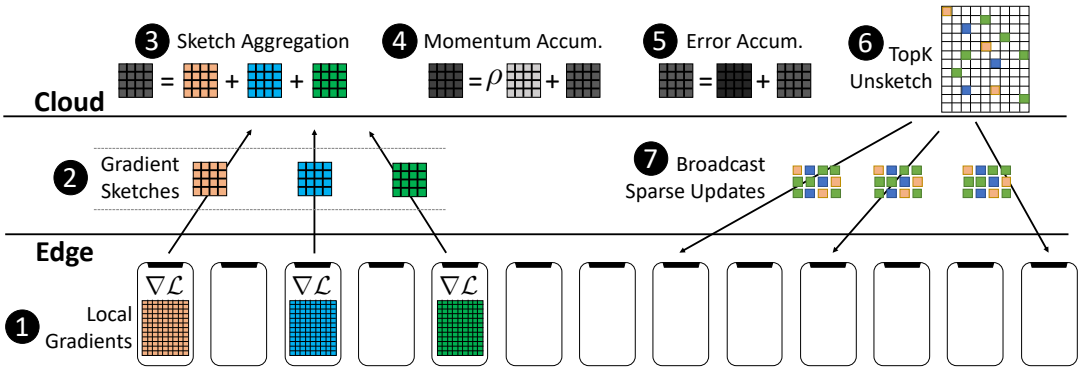


Figure 1. **Algorithm Overview**. The `FetchSGD` algorithm **(1)** computes gradients locally, and then send sketches **(2)** of the gradients to the cloud. In the cloud, gradient sketches are aggregated **(3)**, and then **(4)** momentum and **(5)** error accumulation are applied to the sketch. The approximate top-k values are then **(6)** extracted and **(7)** broadcast as sparse updates to devices participating in next round.

ireddy et al. (2019a) modify the local updates in `FedAvg` to make them point closer to the consensus gradient direction from all clients. They achieve good convergence at the cost of making the clients stateful.

2.2. Gradient Compression

A limitation of `FedAvg` is that, in each communication round, clients must download an entire model and upload an entire model update. Because federated clients are typically on slow and unreliable network connections, this requirement makes training large models with `FedAvg` difficult. Uploading model updates is particularly challenging, since residential Internet connections tend to be asymmetric, with far higher download speeds than upload speeds (Goga and Teixeira, 2012).

An alternative to `FedAvg` that helps address this problem is regular distributed SGD with gradient compression. It is possible to compress stochastic gradients such that the result is still an unbiased estimate of the true gradient, for example by stochastic quantization (Alistarh et al., 2017) or stochastic sparsification (Wangni et al., 2018). However, there is a fundamental tradeoff between increasing compression and increasing the variance of the stochastic gradient, which slows convergence. The requirement that gradients remain unbiased after compression is too stringent, and these methods have had limited empirical success.

Biased gradient compression methods, such as top-k sparsification (Lin et al., 2017) or signSGD (Bernstein et al., 2018), have been more successful in practice. These methods rely, both in theory and in practice, on the ability to locally accumulate the error introduced by the compression scheme, such that the error can be re-introduced the next time the client participates (Karimireddy et al., 2019b). Unfortunately, carrying out error accumulation requires local client state, which is often infeasible in federated learning.

2.3. Optimization with Sketching

This work advances the growing body of research applying sketching techniques to optimization. Jiang et al. (2018) propose using sketches for gradient compression in data center training. Their method achieves empirical success when gradients are sparse, but it has no convergence guarantees, and it achieves little compression on dense gradients (Jiang et al., 2018, §B.3). The method also does not make use of error accumulation, which more recent work has demonstrated is necessary for biased gradient compression schemes to be successful (Karimireddy et al., 2019b). Ivkin et al. (2019a) also propose using sketches for gradient compression in data center training. However, their method requires a second round of communication between the clients and the parameter server, after the first round of transmitting compressed gradients completes. Using a second round is not practical in federated learning, since stragglers would delay completion of the first round, at which point a number of clients that had participated in the first round would no longer be available (Bonawitz et al., 2016). Furthermore, the method in (Ivkin et al., 2019a) requires local client state for both momentum and error accumulation, which is not possible in federated learning. Spring et al. (2019) also propose using sketches for distributed optimization. Their method compresses auxiliary variables such as momentum and per-parameter learning rates, without compressing the gradients themselves. In contrast, our method compresses the gradients, and it does not require any additional communication at all to carry out momentum.

Konecny et al. (2016) propose using sketched updates to achieve communication efficiency in federated learning. However, the family of sketches they use differs from the techniques we propose in this paper: they apply a combination of subsampling, quantization and random rotations.

3. FetchSGD

3.1. Federated Learning Setup

Consider a federated learning scenario with C clients, where the i^{th} client has samples D_i drawn i.i.d. from distinct unknown data distributions $\{\mathcal{P}_i\}$. We do not assume that $\mathcal{P}_i$ are related. Let $\mathcal{L} : \mathcal{W} \times \mathcal{X} \to \mathbb{R}$ be a loss function, where the goal is to minimize the weighted empirical average of client risks:

$$f(\mathrm{w}) = \widehat{\mathbb{E}} f_i(\mathrm{w}) = \frac{1}{\sum_{i=1}^{C} |D_i|} \sum_{i=1}^{C} |D_i| \mathop{\mathbb{E}}_{\mathrm{x} \sim \mathcal{P}_i} \mathcal{L}(\mathrm{w}, \mathrm{x}) \quad (1)$$

Assuming that all clients have an equal number of data points, this simplifies to the empirical average of client risks:

$$f(\mathrm{w}) = \widehat{\mathbb{E}} f_i(\mathrm{w}) = \frac{1}{C} \sum_{i=1}^{C} \mathop{\mathbb{E}}_{\mathrm{x} \sim \mathcal{P}_i} \mathcal{L}(\mathrm{w}, \mathrm{x}). \quad (2)$$

For simplicity of presentation, we consider this unweighted average (eqn. 2), but our theoretical results directly extend to the the more general setting (eqn. 1).

In federated learning, a central aggregator coordinates an iterative optimization procedure to minimize f with respect to w, the parameters of the model. In every iteration, the aggregator chooses W clients uniformly at random,[1] and these clients download the current model, determine how to best update the model based on their local data, and upload a model update to the aggregator. The aggregator then combines these model updates to update the model for the next iteration. Different federated optimization algorithms use different model updates and different aggregation schemes to combine these updates.

3.2. Algorithm

At each iteration in `FetchSGD`, the i^{th} participating client computes a stochastic gradient g_t^i using a batch of (or all of) its local data, then compresses g_t^i using a data structure called a Count Sketch. Each client then sends the sketch $\mathcal{S}(\mathrm{g}_t^i)$ to the aggregator as its model update.

A Count Sketch is a randomized data structure that can compress a vector by randomly projecting it several times to lower dimensional spaces, such that high-magnitude elements can later be approximately recovered. We provide more details on the Count Sketch in Appendix C, but here we treat it simply as a compression operator $\mathcal{S}(\cdot)$, with the special property that it is linear:

$$\mathcal{S}(\mathrm{g}_1 + \mathrm{g}_2) = \mathcal{S}(\mathrm{g}_1) + \mathcal{S}(\mathrm{g}_2).$$

[1] In practice, the clients may not be chosen randomly, since often only devices that are on wifi, charging, and idle are allowed to participate.

Using linearity, the server can exactly compute the sketch of the true minibatch gradient $\mathrm{g}^t = \sum_i \mathrm{g}_i^t$ given only the $\mathcal{S}(\mathrm{g}_i^t)$,

$$\sum_i \mathcal{S}(\mathrm{g}_i^t) = \mathcal{S}\left(\sum_i \mathrm{g}_i^t\right) = \mathcal{S}(\mathrm{g}^t).$$

Another useful property of the Count Sketch is that, for a sketching operator $\mathcal{S}(\cdot)$, there is a corresponding decompression operator $\mathcal{U}(\cdot)$ that returns an unbiased estimate of the original vector, such that the high-magnitude elements of the vector are approximated well (see Appendix C for details):

$$\text{Top-k}(\mathcal{U}(\mathcal{S}(\mathrm{g}))) \approx \text{Top-k}(\mathrm{g}).$$

Briefly, $\mathcal{U}(\cdot)$ approximately "undoes" the projections computed by $\mathcal{S}(\cdot)$ for each row, and then takes a median across rows to reduce the variance of the final estimate. See Appendix C for more details.

With the $\mathcal{S}(\mathrm{g}_i^t)$ in hand, the central aggregator could update the global model with Top-k $\left(\mathcal{U}(\sum_i \mathcal{S}(\mathrm{g}_i^t))\right) \approx$ Top-k $\left(\mathrm{g}^t\right)$. However, Top-k$(\mathrm{g}^t)$ is not an unbiased estimate of g^t, so the normal convergence of SGD does not apply. Fortunately, Karimireddy et al. (2019b) show that biased gradient compression methods can converge if they accumulate the error incurred by the biased gradient compression operator and re-introduce the error later in optimization. In `FetchSGD`, the bias is introduced by Top-k rather than by $\mathcal{S}(\cdot)$, so the aggregator, instead of the clients, can accumulate the error, and it can do so into a zero-initialized sketch S_e instead of into a gradient-like vector:

$$\begin{aligned}
\mathrm{S}^t &= \frac{1}{W} \sum_{i=1}^{W} \mathcal{S}(\mathrm{g}_i^t) \\
\Delta^t &= \text{Top-k}(\mathcal{U}(\eta \mathrm{S}^t + \mathrm{S}_e^t)) \\
\mathrm{S}_e^{t+1} &= \eta \mathrm{S}^t + \mathrm{S}_e^t - \mathcal{S}(\Delta^t) \\
\mathrm{w}^{t+1} &= \mathrm{w}^t - \Delta^t,
\end{aligned}$$

where η is the learning rate.

In contrast, other biased gradient compression methods introduce bias on the clients when compressing the gradients, so the clients themselves must maintain individual error accumulation vectors. This becomes a problem in federated learning, where clients may participate only once, giving the error no chance to be reintroduced in a later round.

Viewed another way, because $\mathcal{S}(\cdot)$ is linear, and because error accumulation consists only of linear operations, carrying out error accumulation on the server within S_e is equivalent to carrying out error accumulation on each client, and uploading sketches of the result to the server. (Computing the model update from the accumulated error is not linear, but only the server does this, whether the error is accumulated

on the clients or on the server.) Taking this a step further, we note that momentum also consists of only linear operations, and so momentum can be equivalently carried out on the clients or on the server. Extending the above equations with momentum yields

$$\begin{aligned} S^t &= \frac{1}{W}\sum_{i=1}^{W} \mathcal{S}(g_i^t) \\ S_u^{t+1} &= \rho S_u^t + S^t \\ \Delta &= \text{Top-k}(\mathcal{U}(\eta S_u^{t+1} + S_e^t))) \\ S_e^{t+1} &= \eta S_u^{t+1} + S_e^t - \mathcal{S}(\Delta) \\ w^{t+1} &= w^t - \Delta. \end{aligned}$$

`FetchSGD` is presented in full in Algorithm 1.

Algorithm 1 `FetchSGD`

Input: number of model weights to update each round k
Input: learning rate η
Input: number of timesteps T
Input: momentum parameter ρ, local batch size ℓ
Input: Client datasets $\{D_i\}_{i=1}^{C}$
Input: Number of clients selected per round W
Input: Loss function $\mathcal{L}(\text{model weights}, \text{datum})$
Input: Sketching and unsketching functions $\mathcal{S}, \mathcal{U}$
Initialize S_u^0 and S_e^0 to zero sketches
Initialize w_0 using the same random seed on the clients and aggregator
for $t = 1, 2, \cdots T$ **do**
  Randomly select W clients $c_1, \ldots c_W$
  loop {In parallel on clients $\{c_i\}_{i=1}^{W}$}
    Download (possibly sparse) new model weights $w^t - w^0$
    Compute stochastic gradient g_i^t on batch B_i of size ℓ: $g_t^i = \frac{1}{\ell}\sum_{j=1}^{l} \nabla_w \mathcal{L}(w^t, x_j)$
    Sketch g_t^i: $S_i^t = \mathcal{S}(g_t^i)$ and send it to the Aggregator
  end loop
  Aggregate sketches $S^t = \frac{1}{W}\sum_{i=1}^{W} S_i^t$
  Momentum: $S_u^t = \rho S_u^{t-1} + S^t$
  Error feedback: $S_e^t = \eta S_u^t + S_e^t$
  Unsketch: $\Delta^t = \text{Top-k}(\mathcal{U}(S_e^t))$
  Error accumulation: $S_e^{t+1} = S_e^t - S(\Delta^t)$
  Update $w_{t+1} = w_t - \Delta^t$
end for
Output: $\{w_t\}_{t=1}^{T}$

4. Theory

This section presents convergence guarantees for `FetchSGD`. First, Section 4.1 gives the convergence of `FetchSGD` when making a strong and opaque assumption about the sequence of gradients. Section 4.2 instead makes a more interpretable assumption about the gradients, and arrives at a weaker convergence guarantee.

4.1. Scenario 1: Contraction Holds

To show that compressed SGD converges when using a biased compression operator, existing methods first show that their compression operator obeys a contraction property, and then they appeal to Stich et al. (2018) for convergence guarantees (Karimireddy et al., 2019b; Zheng et al., 2019; Ivkin et al., 2019a). Specifically, for the convergence results of Stich et al. (2018) to apply, the compression operator $\mathcal{C}$ must be a τ-contraction:

$$\|\mathcal{C}(x) - x\| \leq (1-\tau)\|x\|$$

Ivkin et al. (2019a) show that it is possible to satisfy this contraction property using Count Sketches to compress gradients. However, their compression method includes a second round of communication: if there are no high-magnitude elements in e_t, as computed from $\mathcal{S}(e_t)$, the server can query clients for random entries of e_t. On the other hand, `FetchSGD` never computes the e_t^i, or e_t, so this second round of communication is not possible, and the analysis of Ivkin et al. (2019a) does not apply. In this section, we simply assume that the contraction property holds along the optimization path. Because Count Sketches approximate ℓ_2 norms, we can phrase this assumption in terms of sketched quantities that are actually computed in the algorithm:

Assumption 1 (Scenario 1)**.** *For the sequence of gradients encountered during optimization, there exists a constant $0 < \tau \leq 1$ such that the following holds*

$$\|S(e_t + \eta(g_t + \rho u_{t-1})\|^2 \leq (1-\tau)\|S(\eta(g_t + \rho u_{t-1}))\|^2$$

Theorem 1 (Scenario 1)**.** *Let f be an L-smooth [2] non-convex function and let g_i denote stochastic gradients of f_i such that $\mathbb{E}\|g_i\|^2 \leq G_i^2$, and $G^2 := \frac{\sum_{i=1}^{C} G_i^2}{C}$. Under Assumption 1, `FetchSGD`, with step size $\eta = \frac{1-\rho}{2L\sqrt{T}}$, in T iterations, returns $\{w_t\}_{t=1}^{T}$ such that*

1. $\min_{t=1\cdots T} \mathbb{E}\|f(w_t)\|^2 \leq \frac{4L(f(w_0)-f^*) + (1-\rho)\sigma^2}{\sqrt{T}} + \frac{2(1-\tau)G^2}{\tau^2 T}$
2. *The sketch uploaded from each participating client to the parameter server is $\mathcal{O}\left(k \log\left(dT/\delta\right)\right)$ bytes per round.*

Note that the contraction factor τ should be considered a function of k, highlighting the trade-off between communication and utility.

Intuitively, Assumption 1 states that, at each time step, the descent direction – *i.e.*, the scaled negative gradient, including momentum – and the error accumulation vector must point in sufficiently the same direction. This assumption is rather opaque, since it involves all of the gradient, momentum, and error accumulation vectors, and it is not immediately obvious that we should expect it to hold. To remedy this, the next section analyzes `FetchSGD` under a simpler assumption that involves only the gradients. Note that this is still an assumption on the algorithmic path, but it presents a clearer understanding.

[2] A differentiable function f is L-smooth if $\|\nabla f(x) - \nabla f(y)\| \leq L\|x - y\| \;\; \forall\, x, y \in \text{dom}(f)$.

4.2. Scenario 2: Sliding Window Heavy Hitters

Gradients taken along the optimization path have been observed to contain heavy coordinates (Shi et al., 2019; Li et al., 2019). However, it would be overly optimistic to assume that all gradients contain heavy coordinates. This might break down, for example, in very flat regions of parameter space. Instead, we introduce a much milder assumption: namely that there exist heavy coordinates in a sliding sum of gradient vectors:

Definition 1. *[(I,α)-sliding heavy[3]]*
A stochastic process $\{g_t\}_t$ is (I,α)-sliding heavy if, at any iteration t, the gradient vector g_t can be decomposed as $\mathrm{g}_t = \mathrm{g}_t^N + \mathrm{g}_t^S$, where g_t^S is "signal" and g_t^N is "noise" with the following properties:

1. ***[Signal]** With probability at least $1-\delta$, for every non-zero coordinate j of vector g_t^S: $\exists t_1, t_2$ with $t_1 \leq t \leq t_2$, $t_1 - t_2 \leq I$: $|\sum_{t_1}^{t_2} \mathrm{g}_t^j| > \alpha \|\sum_{t_1}^{t_2} \mathrm{g}_t\|$.*
2. ***[Noise]** g_t^N is mean zero, symmetric and when normalized by its norm, its second moment bounded as $\mathbb{E}\frac{\|\mathrm{g}_t^N\|^2}{\|\mathrm{g}_t\|^2} \leq \beta$.*

Intuitively, this definition states that, if we sum up to I consecutive gradients, every coordinate in the result will either be an α-heavy hitter, or will be drawn from some mean-zero symmetric noise. When $I = 1$, part 1 of the definition reduces to the assumption that gradients always contain heavy coordinates. Our assumption for general, constant I is significantly weaker, as it requires the gradients to have heavy coordinates in a sequence of I iterations rather than in every iteration. The existence of heavy coordinates spread across consecutive updates helps to explains the success of error feedback techniques, which extract signal from a sequence of gradients that may be indistinguishable from noise in any one iteration. Note that both the signal and the noise scale with the norm of the gradient, so both adjust accordingly as gradients become smaller later in optimization.

Under this definition, we can use Count Sketches to capture the signal, since Count Sketches can approximate heavy hitters. Because the signal is spread over sliding windows of size I, we need a sliding window error accumulation scheme to ensure that we capture whatever signal is present. Vanilla error accumulation is not sufficient to show convergence, since vanilla error accumulation sums up *all* prior gradients, so signal that is present only in a sum of I consecutive gradients (but not in $I+1$, or $I+2$, etc.) will not be captured with vanilla error accumulation. Instead, `FetchSGD` uses a sliding window error accumulation scheme, which can capture any signal that is spread over a sequence of at most I gradients. One simple way to accomplish this is to maintain I error accumulation Count Sketches, as shown in Figure 2 for $I = 4$. Each sketch accumulates new gradients every iteration, and beginning at offset iterations, each sketch is zeroed out every I iterations before continuing to accumulate gradients. Under this scheme, at every iteration there is a sketch available that contains the sketched sum of the prior I' gradients, for all $I' \leq I$.

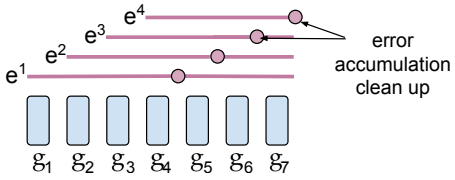


Figure 2. Sliding window error accumulation

In practice, it is too expensive to maintain I error accumulation sketches. Fortunately, this "sliding window" problem is well studied in the sketching community (Datar et al., 2002; Braverman and Ostrovsky, 2007), and it is possible to identify heavy hitters that are spread over a sequence of gradients with only $\log(I)$ error accumulation sketches. Additional details on sliding window Count Sketch are in Appendix D. Although we use a sliding window error accumulation scheme to prove convergence, in all experiments we use a single error accumulation sketch, since we find that doing so still leads to good convergence.

Assumption 2 (Scenario 2)**.** *The sequence of gradients encountered during optimization form an (I,α)-sliding heavy stochastic process.*

Theorem 2 (Scenario 2)**.** *Let f be an L-smooth non-convex function and let g_i denote stochastic gradients of f_i such that $\mathbb{E}\|\mathrm{g}_i\|^2 \leq G_i^2$, and $G^2 := \frac{\sum_{i=1}^C G_i^2}{C}$. Under Assumption 2, `FetchSGD`, with step size $\eta = \frac{1}{G\sqrt{L}T^{2/3}}$ and $\rho = 0$ (no momentum), in T iterations, with probability at least $1-\delta$, returns $\{\mathrm{w}_t\}_{t=1}^T$ such that*

1. $\min_{t=1\cdots T} \mathbb{E}\|\nabla f(\mathrm{w}_t)\|^2 \leq \frac{G\sqrt{L}\left((f(\mathrm{w}_0)-f^*)+2(2-\alpha)+2I^2\right)}{T^{1/3}} + \frac{G\sqrt{L}}{T^{2/3}}$
2. *The sketch uploaded from each participating client to the parameter server is $\mathcal{O}\left(\frac{\log(dT/\delta)}{\alpha^2}\right)$ bytes per round.*

Remarks:

1. These guarantees are for the non-i.i.d. setting – i.e. f is the average risk with respect to potentially unrelated distributions (see eqn. 2).
2. The convergence rate in Theorem 1 matches that of uncompressed SGD, while the rate in Theorem 2 is worse.
3. The proof uses the virtual sequence idea of Stich et al. (2018), and can be generalized to other class of functions like smooth, (strongly) convex etc. by careful averaging (proof in Appendix B.2.2).

[3] Technically, this definition is also parameterized by δ and β. However, in the interest of brevity, we use the simpler term "(I,α)-sliding heavy" throughout the manuscript.

5. Evaluation

We implement and compare FetchSGD, gradient sparsification (local top-k), and FedAvg using PyTorch (Paszke et al., 2019).[4] We note the following differences between the theoretical and empirical algorithms:

- We test on neural networks containing ReLU, whose loss surfaces are not L-smooth.
- Our theory for Scenario 2 uses a sliding window Count Sketch for error accumulation, but in practice we use a vanilla Count Sketch.
- We use non-zero momentum (Theorem 1 allows momentum, but Theorem 2 does not).
- For all methods, we employ momentum factor masking, following Lin et al. (2017).
- On line 14 of Algorithm 1, we zero out the nonzero coordinates of $\mathcal{S}(\Delta^t)$ in S_e^t instead of subtracting $\mathcal{S}(\Delta^t)$. Empirically, doing so stabilizes the optimization.

We focus our experiments on the regime of small local datasets and non-i.i.d. data, since we view this as both an important and relatively unsolved regime in federated learning. Gradient sparsification methods, which sum together the local top-k gradient elements from each worker, do a worse job approximating the true top-k of the global gradient as local datasets get smaller and more unlike each other. And taking many steps on each client's local data, which is how FedAvg achieves communication efficiency, is unproductive since it leads to immediate local overfitting. However, real-world users tend to generate data with sizes that follow a power law distribution (Goyal et al., 2017), so most users will have relatively small local datasets. Real data in the federated setting is also typically non-i.i.d.

FetchSGD has a key advantage over prior methods in this regime because our compression operator is linear. Small local datasets pose no difficulties, since executing a step using only a single client with N data points is equivalent to executing a step using N clients, each of which has only a single data point. By the same argument, issues arising from non-i.i.d. data are partially mitigated by random client selection, since combining the data of participating clients leads to a more representative sample of the full data distribution.

For each method, we report the compression achieved relative to uncompressed SGD in terms of total bytes uploaded and downloaded.[5] One important consideration not captured in these numbers is that in FedAvg, clients must download an entire model immediately before participating, because every model weight could get updated in every round. In contrast, local top-k and FetchSGD only update a limited number of parameters per round, so non-participating clients can stay relatively up to date with the current model, reducing the number of new parameters that must be downloaded immediately before participating. This makes upload compression more important than download compression for local top-k and FetchSGD. Download compression is also less important for all three methods since residential Internet connections tend to reach far higher download than upload speeds (Goga and Teixeira, 2012). We include results here of overall compression (including upload and download), but break up the plots into separate upload and download components in the Appendix, Figure 6.

In all our experiments, we tune standard hyperparameters on the uncompressed runs, and we maintain these same hyperparameters for all compression schemes. Details on which hyperparameters were chosen for each task can be found in Appendix A. FedAvg achieves compression by reducing the number of iterations carried out, so for these runs, we simply scale the learning rate schedule in the iteration dimension to match the total number of iterations that FedAvg will carry out. We report results for each compression method over a range of hyperparameters: for local top-k, we adjust k; and for FetchSGD we adjust k and the number of columns in the sketch (which controls the compression rate of the sketch). We tune the number of local epochs and federated averaging batch size for FedAvg, but do not tune the learning rate decay for FedAvg because we find that FedAvg does not approach the baseline accuracy on our main tasks for even a small number of local epochs, where the learning rate decay has very little effect.

In the non-federated setting, momentum is typically crucial for achieving high performance, but in federating learning, momentum can be difficult to incorporate. Each client could carry out momentum on its local gradients, but this is ineffective when clients participate only once or a few times. Instead, the central aggregator can carry out momentum on the aggregated model updates. For FedAvg and local top-k, we experiment with ($\rho_g = 0.9$) and without ($\rho_g = 0$) this global momentum. For each method, neither choice of ρ_g consistently performs better across our tasks, reflecting the difficulty of incorporating momentum. In contrast, FetchSGD incorporates momentum seamlessly due to the linearity of our compression operator (see Section 3.2); we use a momentum parameter of 0.9 in all experiments.

In all plots of performance vs. compression, each point represents a trained model, and for clarity, we plot only the Pareto frontier over hyperparameters for each method. Figures 7 and 9 in the Appendix show results for all runs that converged.

[4] Code available at https://github.com/kiddyboots216/CommEfficient. Git commit at the time of camera-ready: 833ca44.

[5] We only count non-zero weight updates when computing how many bytes are transmitted. This makes the unrealistic assumption that we have a zero-overhead sparse vector encoding scheme.

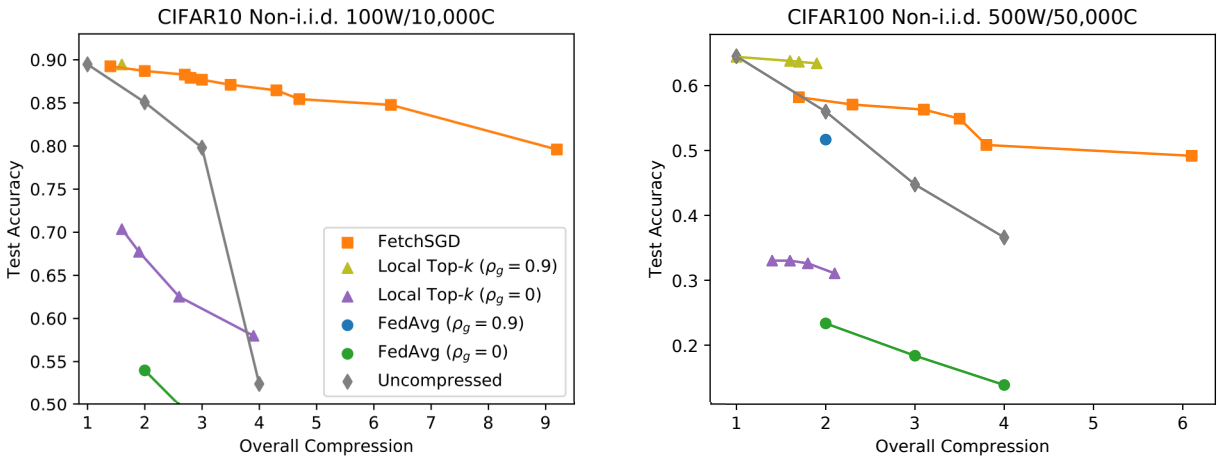


Figure 3. Test accuracy achieved on CIFAR10 (left) and CIFAR100 (right). "Uncompressed" refers to runs that attain compression by simply running for fewer epochs. `FetchSGD` outperforms all methods, especially at higher compression. Many `FedAvg` and local top-k runs are excluded from the plot because they failed to converge or achieved very low accuracy.

5.1. CIFAR (ResNet9)

CIFAR10 and CIFAR100 (Krizhevsky et al., 2009) are image classification datasets with 60,000 32×32 pixel color images distributed evenly over 10 and 100 classes respectively (50,000/10,000 train/test split). They are benchmark datasets for computer vision, and although they do not have a natural non-i.i.d. partitioning, we artificially create one by giving each client images from only a single class. For CIFAR10 (CIFAR100) we use 10,000 (50,000) clients, yielding 5 (1) images per client. Our 7M-parameter model architecture, data preprocessing, and most hyperparameters follow Page (2019), with details in Appendix A.1. We report accuracy on the test datasets.

Figure 3 shows test accuracy vs. compression for CIFAR10 and CIFAR100. In this setting with very small local datasets, `FedAvg` and local top-k both struggle to achieve significantly better results than uncompressed SGD. Although we ran a large hyperparameter sweep, many runs simply diverge, especially for higher compression (local top-k) or more local iterations (`FedAvg`). We expect this setting to be challenging for `FedAvg`, since running multiple gradient steps on only one or a few data points, especially points that are not representative of the overall distribution, is unlikely to be productive. And although local top-k can achieve high upload compression, download compression is reduced to almost $1\times$, since summing sparse gradients from many workers, each with very different data, leads to a nearly dense model update each round.

5.2. FEMNIST (ResNet101)

The experiments above show that `FetchSGD` significantly outperforms competing methods in the regime of very small local datasets and non-i.i.d. data. In this section we introduce a task designed to be more favorable for `FedAvg`, and show that `FetchSGD` still performs competitively.

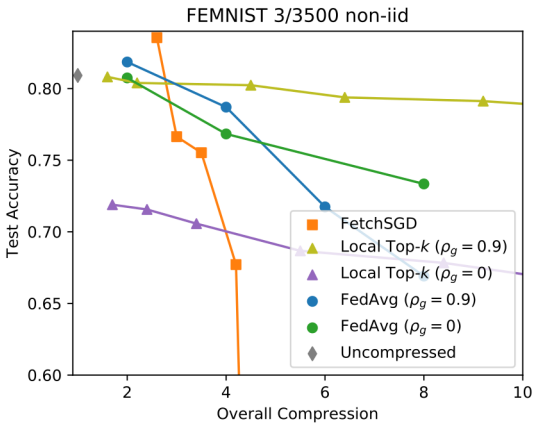


Figure 4. Test accuracy on FEMNIST. The dataset is not very non-i.i.d., and has relatively large local datasets, but `FetchSGD` is still competitive with `FedAvg` and local top-k for lower compression.

Federated EMNIST is an image classification dataset with 62 classes (upper- and lower-case letters, plus digits) (Caldas et al., 2018), which is formed by partitioning the EMNIST dataset (Cohen et al., 2017) such that each client in FEMNIST contains characters written by a particular person. Experimental details, including our 40M-parameter model architecture, can be found Appendix A.2. We report the final accuracy of the trained models on the validation dataset. The baseline run trains for a single epoch (*i.e.*, each client participates once).

FEMNIST was introduced as a benchmark dataset for `FedAvg`, and it has relatively large local dataset sizes ($\sim$ 200 images per client). The clients are split accord-

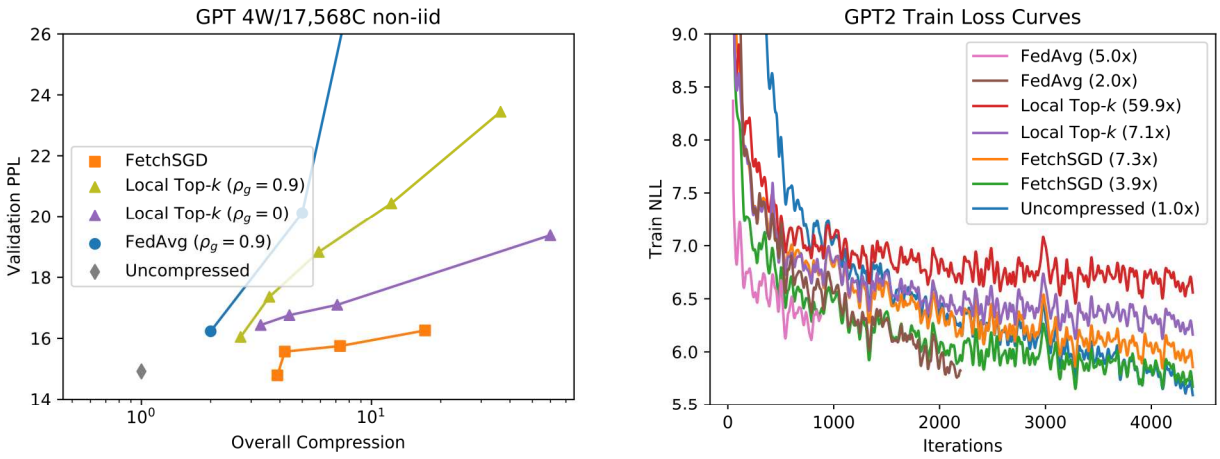


Figure 5. Left: Validation perplexity achieved by finetuning GPT2-small on PersonaChat. `FetchSGD` achieves 3.9× compression without loss in accuracy over uncompressed SGD, and it consistently achieves lower perplexity than `FedAvg` and top-k runs with similar compression. Right: Training loss curves for representative runs. Global momentum hinders local top-k in this case, so local top-k runs with $\rho_g = 0.9$ are omitted here to increase legibility.

ing to the person who wrote the character, so the data is less non-i.i.d. than our per-class splits of CIFAR10. To maintain a reasonable overall batch size, only three clients participate each round, reducing the need for a linear compression operator. Despite this, `FetchSGD` performs competitively with both `FedAvg` and local top-k for some compression values, as shown in Figure 4.

For low compression, `FetchSGD` actually outperforms the uncompressed baseline, likely because updating only k parameters per round regularizes the model. Interestingly, local top-k using global momentum significantly outperforms other methods on this task, though we are not aware of prior work suggesting this method for federated learning. Despite this surprising observation, local top-k with global momentum suffers from divergence and low accuracy on our other tasks, and it lacks any theoretical guarantees.

5.3. PersonaChat (GPT2)

In this section we consider GPT2-small (Radford et al., 2019), a transformer model with 124M parameters that is used for language modeling. We finetune a pretrained GPT2 on the PersonaChat dataset, a chit-chat dataset consisting of conversations between Amazon Mechanical Turk workers who were assigned faux personalities to act out (Zhang et al., 2018). The dataset has a natural non-i.i.d. partitioning into 17,568 clients based on the personality that was assigned. Our experimental procedure follows Wolf (2019). The baseline model trains for a single epoch, meaning that no local state is possible, and we report the final perplexity (a standard metric for language models; lower is better) on the validation dataset in Figure 5.

Figure 5 also plots loss curves (negative log likelihood) achieved during training for some representative runs. Somewhat surprisingly, all the compression techniques outperform the uncompressed baseline early in training, but most saturate too early, when the error introduced by the compression starts to hinder training.

Sketching outperforms local top-k for all but the highest levels of compression, because local top-k relies on local state for error feedback, which is impossible in this setting. We expect this setting to be challenging for `FedAvg`, since running multiple gradient steps on a single conversation which is not representative of the overall distribution is unlikely to be productive.

6. Discussion

Federated learning has seen a great deal of research interest recently, particularly in the domain of communication efficiency. A considerable amount of prior work focuses on decreasing the total number of communication rounds required to converge, without reducing the communication required in each round. In this work, we complement this body of work by introducing `FetchSGD`, an algorithm that reduces the amount of communication required each round, while still conforming to the other constraints of the federated setting. We particularly want to emphasize that `FetchSGD` easily addresses the setting of non-i.i.d. data, which often complicates other methods. The optimal algorithm for many federated learning settings will no doubt combine efficiency in number of rounds and efficiency within each round, and we leave an investigation into optimal ways of combining these approaches to future work.

Acknowledgements

This research was supported in part by NSF BIGDATA awards IIS-1546482, IIS-1838139, NSF CAREER award IIS-1943251, NSF CAREER grant 1652257, NSF GRFP grant DGE 1752814, ONR Award N00014-18-1-2364 and the Lifelong Learning Machines program from DARPA/MTO. RA would like to acknowledge support provided by Institute for Advanced Study.

In addition to NSF CISE Expeditions Award CCF-1730628, this research is supported by gifts from Alibaba, Amazon Web Services, Ant Financial, CapitalOne, Ericsson, Facebook, Futurewei, Google, Intel, Microsoft, Nvidia, Scotiabank, Splunk and VMware.

References

Dan Alistarh, Demjan Grubic, Jerry Li, Ryota Tomioka, and Milan Vojnovic. Qsgd: Communication-efficient sgd via gradient quantization and encoding. In *Advances in Neural Information Processing Systems*, pages 1709–1720, 2017.

Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. *Journal of Computer and system sciences*, 58(1):137–147, 1999.

Eugene Bagdasaryan, Andreas Veit, Yiqing Hua, Deborah Estrin, and Vitaly Shmatikov. How to backdoor federated learning, 2018.

Jeremy Bernstein, Yu-Xiang Wang, Kamyar Azizzadenesheli, and Anima Anandkumar. signsgd: Compressed optimisation for non-convex problems. *arXiv preprint arXiv:1802.04434*, 2018.

Arjun Nitin Bhagoji, Supriyo Chakraborty, Prateek Mittal, and Seraphin Calo. Analyzing federated learning through an adversarial lens. *arXiv preprint arXiv:1811.12470*, 2018.

Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for federated learning on user-held data. *arXiv preprint arXiv:1611.04482*, 2016.

Vladimir Braverman and Rafail Ostrovsky. Smooth histograms for sliding windows. In *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS'07)*, pages 283–293. IEEE, 2007.

Vladimir Braverman, Stephen R Chestnut, Nikita Ivkin, Jelani Nelson, Zhengyu Wang, and David P Woodruff. Bptree: an ℓ_2 heavy hitters algorithm using constant memory. In *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 361–376, 2017.

Theodora S Brisimi, Ruidi Chen, Theofanie Mela, Alex Olshevsky, Ioannis Ch Paschalidis, and Wei Shi. Federated learning of predictive models from federated electronic health records. *International journal of medical informatics*, 112:59–67, 2018.

Sebastian Caldas, Sai Meher Karthik Duddu, Peter Wu, Tian Li, Jakub Konecny, H. Brendan McMahan, Virginia Smith, and Ameet Talwalkar. Leaf: A benchmark for federated settings, 2018.

Moses Charikar, Kevin Chen, and Martin Farach-Colton. Finding frequent items in data streams. In *International Colloquium on Automata, Languages, and Programming*, pages 693–703. Springer, 2002.

G. Cohen, S. Afshar, J. Tapson, and A. van Schaik. Emnist: Extending mnist to handwritten letters. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 2921–2926, May 2017. doi: 10.1109/IJCNN.2017.7966217.

Mayur Datar, Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Maintaining stream statistics over sliding windows. *SIAM journal on computing*, 31(6):1794–1813, 2002.

Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Marc'aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, et al. Large scale distributed deep networks. In *Advances in neural information processing systems*, pages 1223–1231, 2012.

EU. 2018 reform of eu data protection rules, 2018. URL https://tinyurl.com/ydaltt5g.

Robin C. Geyer, Tassilo Klein, and Moin Nabi. Differentially private federated learning: A client level perspective, 2017.

Oana Goga and Renata Teixeira. Speed measurements of residential internet access. In *International Conference on Passive and Active Network Measurement*, pages 168–178. Springer, 2012.

Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

Andrew Hard, Kanishka Rao, Rajiv Mathews, Swaroop Ramaswamy, Francoise Beaufays, Sean Augenstein, Hubert Eichner, Chloé Kiddon, and Daniel Ramage. Federated

learning for mobile keyboard prediction. *arXiv preprint arXiv:1811.03604*, 2018.

Stephen Hardy, Wilko Henecka, Hamish Ivey-Law, Richard Nock, Giorgio Patrini, Guillaume Smith, and Brian Thorne. Private federated learning on vertically partitioned data via entity resolution and additively homomorphic encryption. *arXiv preprint arXiv:1711.10677*, 2017.

Nikita Ivkin, Zaoxing Liu, Lin F Yang, Srinivas Suresh Kumar, Gerard Lemson, Mark Neyrinck, Alexander S Szalay, Vladimir Braverman, and Tamas Budavari. Scalable streaming tools for analyzing n-body simulations: Finding halos and investigating excursion sets in one pass. *Astronomy and computing*, 23:166–179, 2018.

Nikita Ivkin, Daniel Rothchild, Enayat Ullah, Vladimir Braverman, Ion Stoica, and Raman Arora. Communication-efficient distributed sgd with sketching. In *Advances in Neural Information Processing Systems*, pages 13144–13154, 2019a.

Nikita Ivkin, Zhuolong Yu, Vladimir Braverman, and Xin Jin. Qpipe: Quantiles sketch fully in the data plane. In *Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies*, pages 285–291, 2019b.

Jiawei Jiang, Fangcheng Fu, Tong Yang, and Bin Cui. Sketchml: Accelerating distributed machine learning with data sketches. In *Proceedings of the 2018 International Conference on Management of Data*, pages 1269–1284, 2018.

Peter Kairouz, H. Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Keith Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, Rafael G. L. D'Oliveira, Salim El Rouayheb, David Evans, Josh Gardner, Zachary Garrett, Adrià Gascón, Badih Ghazi, Phillip B. Gibbons, Marco Gruteser, Zaid Harchaoui, Chaoyang He, Lie He, Zhouyuan Huo, Ben Hutchinson, Justin Hsu, Martin Jaggi, Tara Javidi, Gauri Joshi, Mikhail Khodak, Jakub Konecny, Aleksandra Korolova, Farinaz Koushanfar, Sanmi Koyejo, Tancrède Lepoint, Yang Liu, Prateek Mittal, Mehryar Mohri, Richard Nock, Ayfer Özgür, Rasmus Pagh, Mariana Raykova, Hang Qi, Daniel Ramage, Ramesh Raskar, Dawn Song, Weikang Song, Sebastian U. Stich, Ziteng Sun, Ananda Theertha Suresh, Florian Tramèr, Praneeth Vepakomma, Jianyu Wang, Li Xiong, Zheng Xu, Qiang Yang, Felix X. Yu, Han Yu, and Sen Zhao. Advances and open problems in federated learning, 2019.

Sai Praneeth Karimireddy, Satyen Kale, Mehryar Mohri, Sashank J. Reddi, Sebastian U. Stich, and Ananda Theertha Suresh. Scaffold: Stochastic controlled averaging for on-device federated learning, 2019a.

Sai Praneeth Karimireddy, Quentin Rebjock, Sebastian U Stich, and Martin Jaggi. Error feedback fixes signsgd and other gradient compression schemes. *arXiv preprint arXiv:1901.09847*, 2019b.

Jakub Konecny, H. Brendan McMahan, Felix X. Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. Federated learning: Strategies for improving communication efficiency, 2016.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. *Master's thesis, Department of Computer Science, University of Toronto*, 2009.

Kyunghan Lee, Joohyun Lee, Yung Yi, Injong Rhee, and Song Chong. Mobile data offloading: How much can wifi deliver? In *Proceedings of the 6th International Conference*, pages 1–12, 2010.

David Leroy, Alice Coucke, Thibaut Lavril, Thibault Gisselbrecht, and Joseph Dureau. Federated learning for keyword spotting. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6341–6345. IEEE, 2019.

He Li, Kaoru Ota, and Mianxiong Dong. Learning iot in edge: Deep learning for the internet of things with edge computing. *IEEE network*, 32(1):96–101, 2018.

Tian Li, Zaoxing Liu, Vyas Sekar, and Virginia Smith. Privacy for free: Communication-efficient learning with differential privacy using sketches. *arXiv preprint arXiv:1911.00972*, 2019.

Yujun Lin, Song Han, Huizi Mao, Yu Wang, and William J Dally. Deep gradient compression: Reducing the communication bandwidth for distributed training. *arXiv preprint arXiv:1712.01887*, 2017.

Zaoxing Liu, Nikita Ivkin, Lin Yang, Mark Neyrinck, Gerard Lemson, Alexander Szalay, Vladimir Braverman, Tamas Budavari, Randal Burns, and Xin Wang. Streaming algorithms for halo finders. In *2015 IEEE 11th International Conference on e-Science*, pages 342–351. IEEE, 2015.

H Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, et al. Communication-efficient learning of deep networks from decentralized data. *arXiv preprint arXiv:1602.05629*, 2016.

Jayadev Misra and David Gries. Finding repeated elements. *Science of computer programming*, 2(2):143–152, 1982.

Lev Muchnik, Sen Pei, Lucas C Parra, Saulo DS Reis, José S Andrade Jr, Shlomo Havlin, and Hernán A Makse. Origins of power-law degree distribution in the heterogeneity of human activity in social networks. *Scientific reports*, 3 (1):1–8, 2013.

Shanmugavelayutham Muthukrishnan et al. Data streams: Algorithms and applications. *Foundations and Trends® in Theoretical Computer Science*, 1(2):117–236, 2005.

David Page. How to train your resnet, Nov 2019. URL https://myrtle.ai/how-to-train-your-resnet/.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlché Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8):9, 2019.

Anit Kumar Sahu, Tian Li, Maziar Sanjabi, Manzil Zaheer, Ameet Talwalkar, and Virginia Smith. On the convergence of federated optimization in heterogeneous networks. *arXiv preprint arXiv:1812.06127*, 2018.

Shaohuai Shi, Xiaowen Chu, Ka Chun Cheung, and Simon See. Understanding top-k sparsification in distributed deep learning. *arXiv preprint arXiv:1911.08772*, 2019.

Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. Edge computing: Vision and challenges. *IEEE internet of things journal*, 3(5):637–646, 2016.

Ryan Spring, Anastasios Kyrillidis, Vijai Mohan, and Anshumali Shrivastava. Compressing gradient optimizers via count-sketches. *arXiv preprint arXiv:1902.00179*, 2019.

Sebastian U Stich, Jean-Baptiste Cordonnier, and Martin Jaggi. Sparsified sgd with memory. In *Advances in Neural Information Processing Systems*, pages 4447–4458, 2018.

Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147, 2013.

Mark Tomlinson, Wesley Solomon, Yages Singh, Tanya Doherty, Mickey Chopra, Petrida Ijumba, Alexander C Tsai, and Debra Jackson. The use of mobile phones as a data collection tool: a report from a household survey in south africa. *BMC medical informatics and decision making*, 9(1):51, 2009.

Jianyu Wang and Gauri Joshi. Cooperative sgd: A unified framework for the design and analysis of communication-efficient sgd algorithms, 2018.

Shiqiang Wang, Tiffany Tuor, Theodoros Salonidis, Kin K Leung, Christian Makaya, Ting He, and Kevin Chan. Adaptive federated learning in resource constrained edge computing systems. *IEEE Journal on Selected Areas in Communications*, 37(6):1205–1221, 2019.

Jianqiao Wangni, Jialei Wang, Ji Liu, and Tong Zhang. Gradient sparsification for communication-efficient distributed optimization. In *Advances in Neural Information Processing Systems*, pages 1299–1309, 2018.

Thomas Wolf. How to build a state-of-the-art conversational ai with transfer learning, May 2019. URL https://tinyurl.com/ryehjbt.

Thomas Wolf, L Debut, V Sanh, J Chaumond, C Delangue, A Moi, P Cistac, T Rault, R Louf, M Funtowicz, et al. Huggingface's transformers: State-of-the-art natural language processing. *ArXiv, abs/1910.03771*, 2019.

Laurence T Yang, BW Augustinus, Jianhua Ma, Ling Tan, and Bala Srinivasan. *Mobile intelligence*, volume 69. Wiley Online Library, 2010.

Timothy Yang, Galen Andrew, Hubert Eichner, Haicheng Sun, Wei Li, Nicholas Kong, Daniel Ramage, and Francoise Beaufays. Applied federated learning: Improving google keyboard query suggestions. *arXiv preprint arXiv:1812.02903*, 2018.

Saizheng Zhang, Emily Dinan, Jack Urbanek, Arthur Szlam, Douwe Kiela, and Jason Weston. Personalizing dialogue agents: I have a dog, do you have pets too?, 2018.

Yue Zhao, Meng Li, Liangzhen Lai, Naveen Suda, Damon Civin, and Vikas Chandra. Federated learning with non-iid data, 2018.

Shuai Zheng, Ziyue Huang, and James Kwok. Communication-efficient distributed blockwise momentum sgd with error-feedback. In *Advances in Neural Information Processing Systems*, pages 11446–11456, 2019.

The Appendix is organized as follows:

- Appendix A lists hyperparameters and model architectures used in all experiments, and includes plots with additional experimental data, including results broken down into upload, download and overall compression.
- Appendix B gives full proofs of convergence for `FetchSGD`.
- Appendix C describes the Count Sketch data structure and how it is used in `FetchSGD`.
- Appendix D provides the high level idea of the sliding window model and describes how to extend a sketch data structure to the sliding window setting.

A. Experimental Details

We run all experiments on commercially available NVIDIA Pascal, Volta and Turing architecture GPUs.

A.1. CIFAR

In all non-`FedAvg` experiments we train for 24 epochs, with 1% of clients participating each round, for a total of 2400 iterations. We use standard train/test splits of 50000 training datapoints and 10000 validation. We use a triangular learning rate schedule which peaks at epoch 5. We use the maximum peak learning rate for which the uncompressed runs converge: 0.3 for CIFAR10, and 0.2 for CIFAR100. We use this learning rate schedule for all compressed runs. `FedAvg` runs for fewer than 24 epochs, so we compress the learning rate schedule in the iteration dimension accordingly. We do not tune the learning rate separately for any of the compressed runs.

We split the datasets into 10,000 (CIFAR10) and 50,000 (CIFAR100) clients, each of which has 5 (CIFAR10) and 1 (CIFAR100) data point(s) from a single target class. In each round, 1% of clients participate, leading to a total batch size of 500 for both datasets (100 clients with 5 data points for CIFAR10, and 500 clients with 1 data point for CIFAR100). We augment the data during training with random crops and random horizontal flips, and we normalize the images by the dataset mean and standard deviation during training and testing. We use a modified ResNet9 architecture with 6.5M parameters for CIFAR10, and 6.6M parameters for CIFAR100. We do not use batch normalization in any experiments, since it is ineffective with the very small local batch sizes we use. Most of these training procedures, and the modified ResNet9 architecture we use, are drawn from the work of Page (2019).

`FetchSGD`, `FedAvg` and local top-k each have unique hyperparameters that we search over. For `FetchSGD`, we try a grid of values for k and the number of columns in the sketch. For k we try values of [10, 25, 50, 75, 100] $\times 10^3$. For the number of columns we try values of [325, 650, 1300, 2000, 3000] $\times 10^3$. We also tune k for local top-k, trying values of [325, 650, 1300, 2000, 3000, 5000] $\times 10^3$. We present results for local top-k with and without global momentum, but not with local momentum: with such a low participation rate, we observe anecdotally that local momentum performs poorly, since the momentum is always stale, and maintaining local momentum and error accumulation vectors for the large number of clients we experiment with is computationally expensive. The two hyperparameters of interest in `FedAvg` are the total number of global epochs to run (which determines the compression), and the number of local epochs to perform. We run a grid search over global epochs of [6, 8, 12] (corresponding to $4\times$, $3\times$, and $2\times$ compression), and local epochs of [2,3,5].

Figure 6 shows the Pareto frontier of results with each method for CIFAR10 and CIFAR100 broken down into upload, download, and overall compression. Figure 7 shows all runs that converged for the two datasets. For CIFAR10, 1 `FetchSGD` run, 3 local top-k runs, and all `FedAvg` runs using global momentum diverged. For CIFAR100, 1 local top-k run and all `FedAvg` runs using global momentum diverged.

A.2. FEMNIST

The dataset consists of 805,263 28×28 pixel grayscale images distributed unevenly over 3,550 classes/users, with an average of 226.83 datapoints per user and standard deviation of 88.94. We further pre-process the data using the preprocessing script provided by the LEAF repository, using the command: `./preprocess.sh -s niid --sf 1.0 -k 0 -t sample`. This results in 706,057 training samples and 80,182 validation samples over 3,500 clients. [6]

[6] Leaf repository: https://tinyurl.com/u2w3twe

We train a 40M-parameter ResNet101 with layer norm instead of batch norm, using an average batch size of ≈ 600 (but varying depending on which clients participate) with standard data augmentation via image transformations and a triangular learning rate schedule. When we train for 1 epoch, the pivot epoch of the learning rate schedule is 0.2, and the peak learning rate is 0.01. When we train for fewer epochs in `FedAvg`, we compress the learning rate schedule accordingly.

For `FetchSGD` we grid-search values for k and the number of columns. For `FetchSGD` we search over k in [50, 100, 200] $\times 10^3$. and the number of sketch columns in [1, 2, 5, 10] $\times 10^6$. For local top-k we search over k in [10, 20, 50, 100, 200, 500, 1000, 2000, 5000, 10000, 20000] $\times 10^3$. We do not use local momentum for local top-k, since each client only participates once. For `FedAvg`, we search over the number of global epochs in [0.125, 0.1667, 0.25, 0.5], the number of local epochs in [1,2,5], and the local batch size in [10,20,50]. Figure 8 shows the Pareto frontier of results for each method, broken down into upload, download, and overall compression. Figure 9 shows all results that converged.

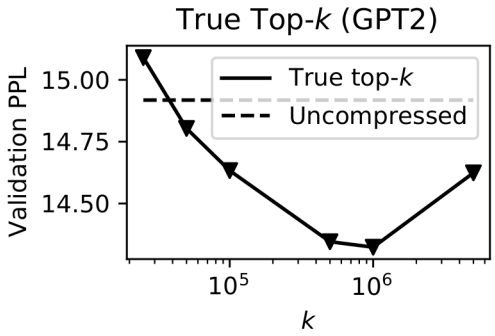


Figure 10. Validation perplexity on PersonaChat for a range of k using true top-k. For $k \approx 10^6$, true top-k provides some regularization, increasing performance over the uncompressed baseline. For larger k, the use of momentum factor masking degrades performance.

A.3. PersonaChat

The non-i.i.d. nature of PersonaChat comes from the fact that different Mechanical Turk workers were provided with different "personalities," which are short snippets, written in English, containing a few salient characteristics of a fictional character. We preprocess the dataset by creating additional tokens denoting the persona, context, and history, and feed these as input to a 124M-parameter GPT2 (Radford et al., 2019) model created by HuggingFace (Wolf et al., 2019) based on the Generative Pretrained Transformer architecture proposed by OpenAI (Radford et al., 2019). We further augment the PersonaChat dataset by randomly shuffling the order of the personality sentences, doubling the size of the local datasets.

We use a linearly decaying learning rate of 0.16, with a total minibatch size of ≈ 64 including the personality augmentation. This can vary depending on which workers participate, as the local datasets are unbalanced.

`FetchSGD`, `FedAvg` and local top-k each have unique hyperparameters which we need to search over. For `FetchSGD` we try 6 points in a grid of values for k and the number of columns. For `FetchSGD`, we search over k in [10, 25, 50, 100, 200] $\times 10^3$, and over the number of sketch columns in [1240, 12400] $\times 10^3$. For local top-k, we search over k in [50, 200, 1240, 5000] $\times 10^3$. For `FedAvg`, we search over the number of global epochs in [0.1, 0.2, 0.5] (10×, 5×, and 2× compression) and the number of local epochs in [2,5,10]. We always use the entire local dataset for each local iteration.

We report the perplexity, which is the average per word branching factor, a standard metric for language models. Although we use the experimental setup and model from Wolf et al. (2019), our perplexities cannot be directly compared due to the modifications made to the choice of optimizer, learning rate, and dataset augmentation strategy. Table 1 shows perplexities, with standard deviations over three runs, for representative runs for each compression method. Learning curves for these runs are shown in Figure 5. Local top-k consistently performs worse on this task when using global momentum (see Figure 5), so we only include results without momentum. Local momentum is not possible, since each client participates only once.

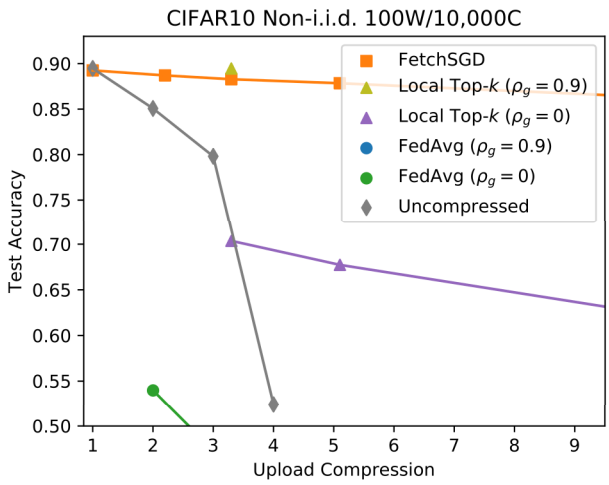
Plots of perplexity vs. compression, broken down into upload, download, and overall compression, can be found in Figure 8.

We note that `FetchSGD` approximates an algorithm where clients send their full gradients, and the server sums those gradients but only updates the model with the k highest-magnitude elements, saving the remaining elements in an error

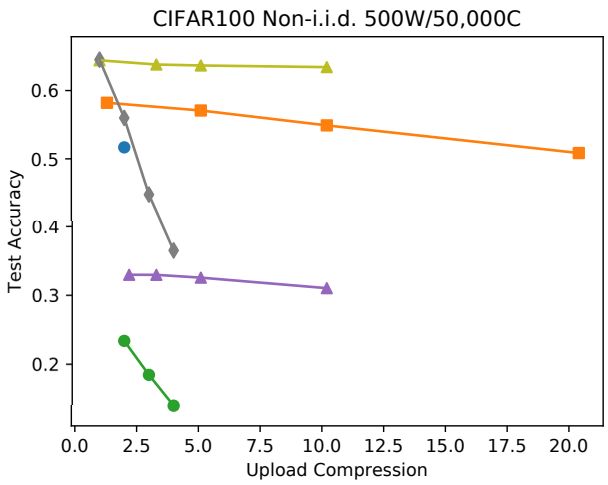
accumulation vector. We explore this method, called true top-k, briefly in Figure 10, which shows the method's performance as a function of k. For intermediate values of k, true top-k actually out-performs the uncompressed baseline, likely because it provides some regularization. For large k, performance reduces because momentum factor masking inhibits momentum.

Method	k	PPL	Download Compression	Upload Compression	Total Compression
Uncompressed	–	14.9 ± 0.02	$1\times$	$1\times$	$1\times$
Local Top-k	50,000	19.3 ± 0.05	$30.3\times$	$2490\times$	$60\times$
Local Top-k	500,000	17.1 ± 0.02	$3.6\times$	$248\times$	$7.1\times$
FedAvg (2 local iters)	–	16.3 ± 0.2	$2\times$	$2\times$	$2\times$
FedAvg (5 local iters)	–	20.1 ± 0.02	$5\times$	$5\times$	$5\times$
Sketch (1.24M cols)	25,000	$\mathbf{15.8 \pm 0.007}$	$3.8\times$	$100\times$	$7.3\times$
Sketch (12.4M cols)	50,000	$\mathbf{14.8 \pm 0.002}$	$2.4\times$	$10\times$	$3.9\times$

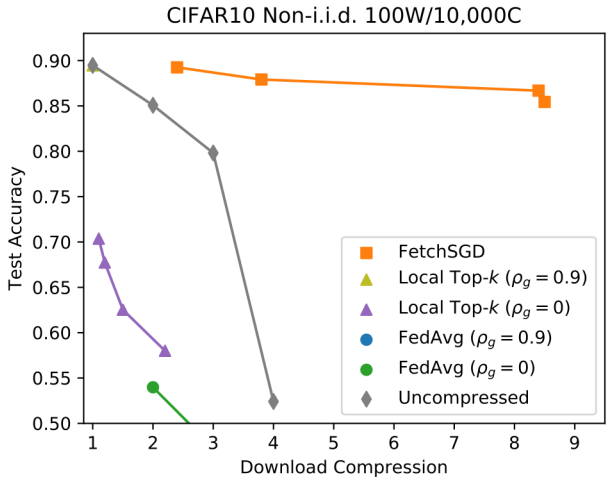
Table 1. Validation perplexities, with standard deviations measured over three different random seeds, for representative runs with `FetchSGD`, local top-k, and `FedAvg` on GPT2. Loss curves for these hyperparameter settings can be found in Figure 5.



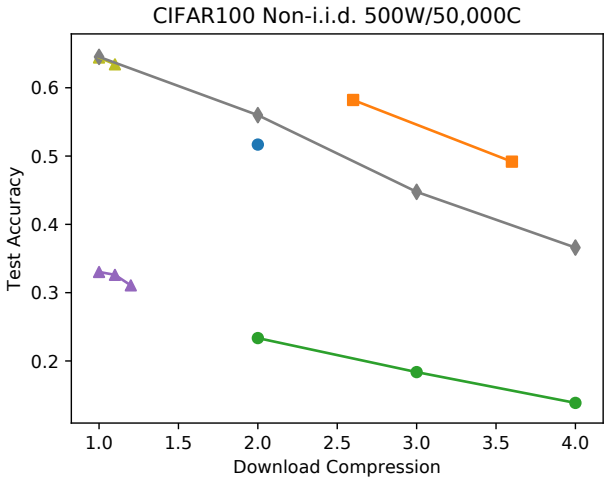
(a) CIFAR10 Upload Compression



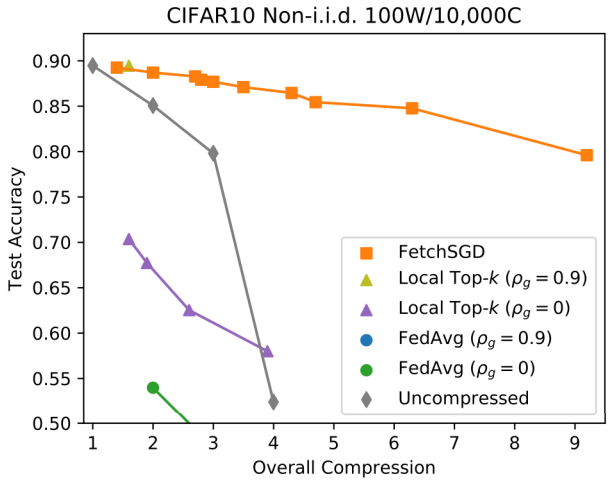
(b) CIFAR100 Upload Compression



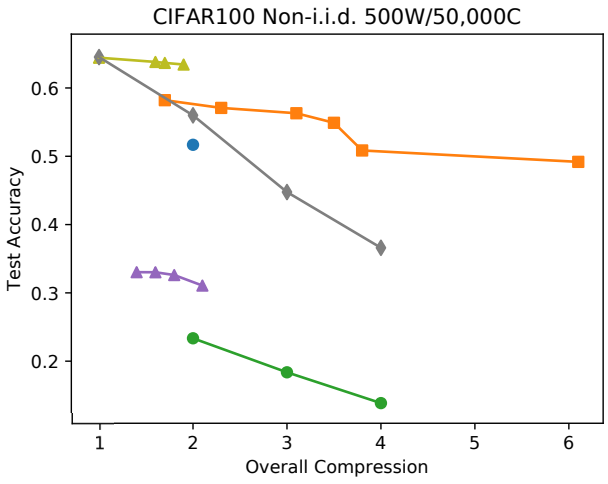
(c) CIFAR10 Download Compression



(d) CIFAR100 Download Compression

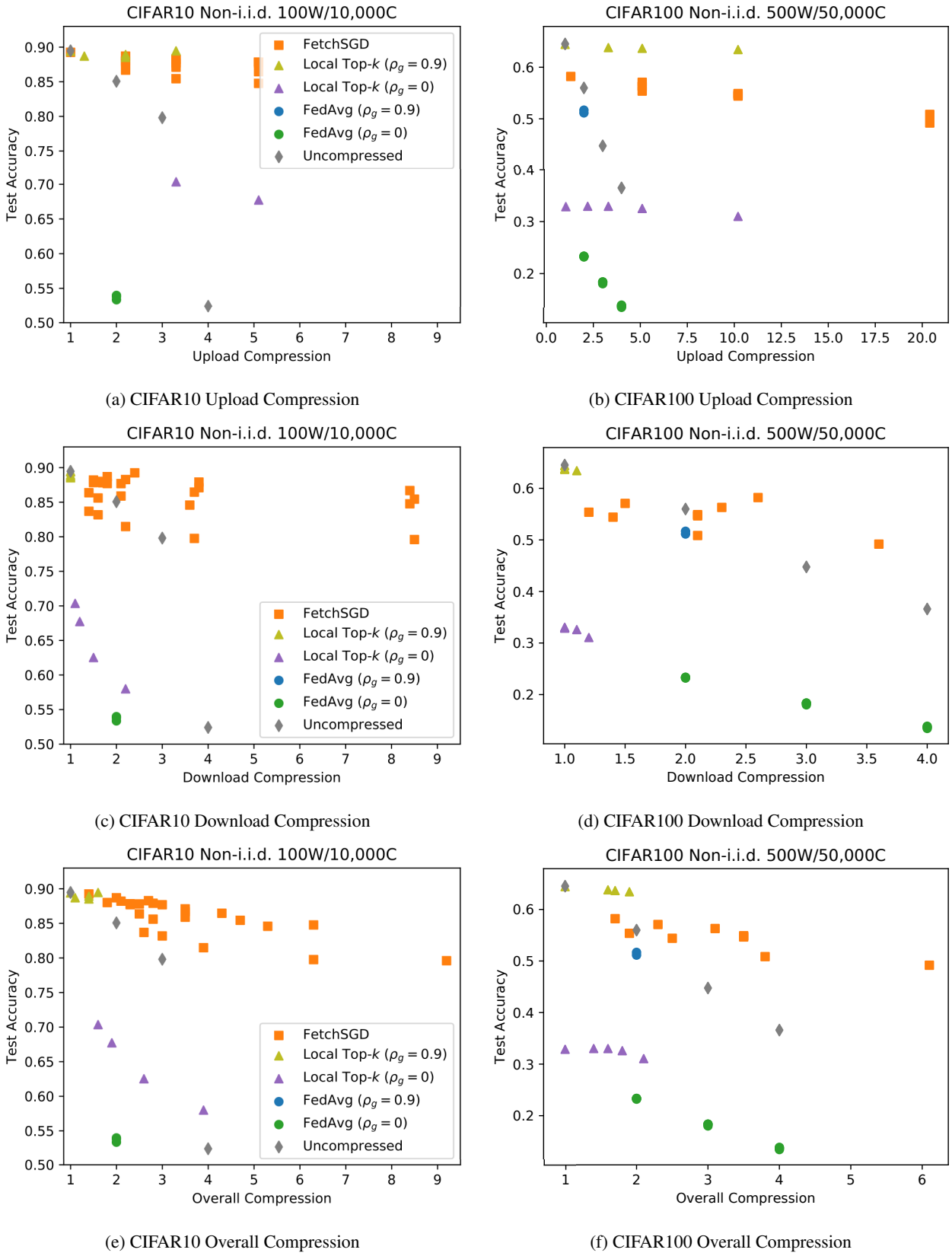


(e) CIFAR10 Overall Compression



(f) CIFAR100 Overall Compression

Figure 6. Upload (top), download (middle), and overall (bottom) compression for CIFAR10 (left) and CIFAR100 (right). To increase readability, each plot shows only the Pareto frontier of runs for the compression type shown in that plot. All runs that converged are shown in Figure 7.



(a) CIFAR10 Upload Compression

(b) CIFAR100 Upload Compression

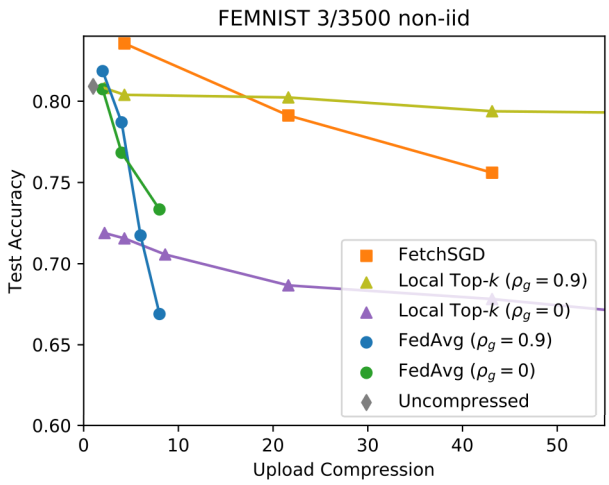
(c) CIFAR10 Download Compression

(d) CIFAR100 Download Compression

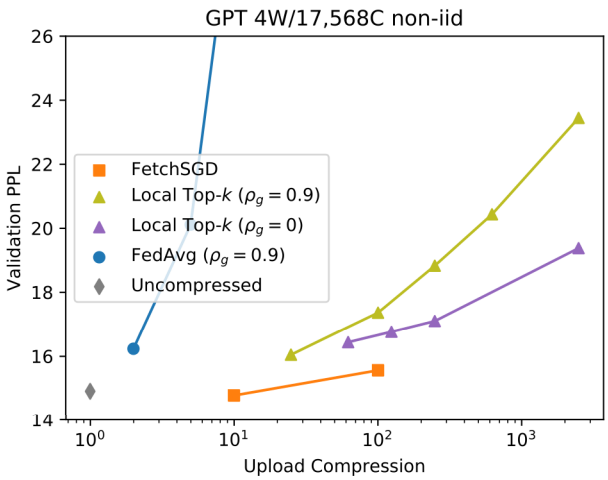
(e) CIFAR10 Overall Compression

(f) CIFAR100 Overall Compression

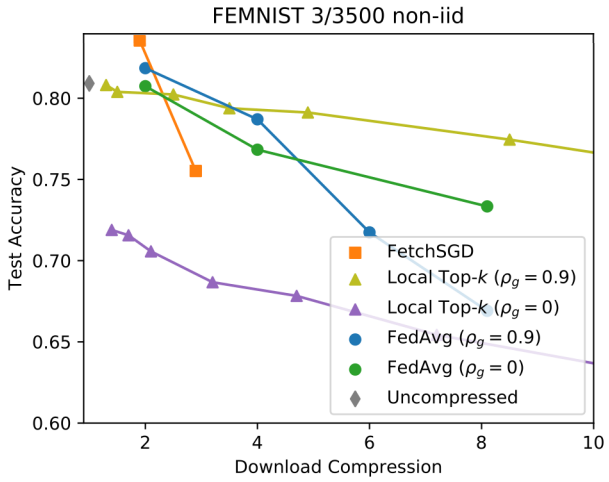
Figure 7. Upload (top), download (middle), and overall (bottom) compression for CIFAR10 (left) and CIFAR100 (right).



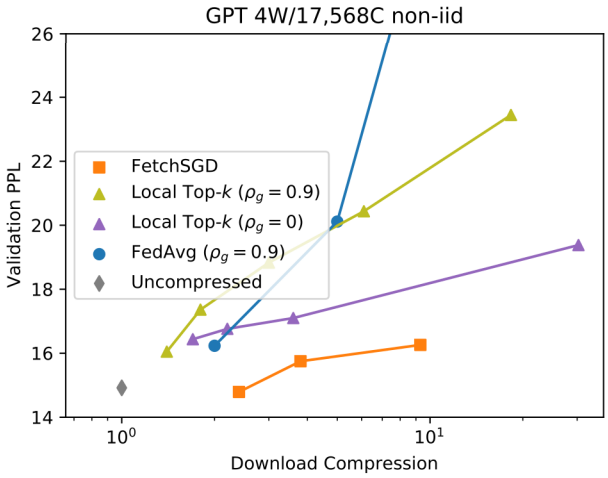
(a) FEMNIST Upload Compression



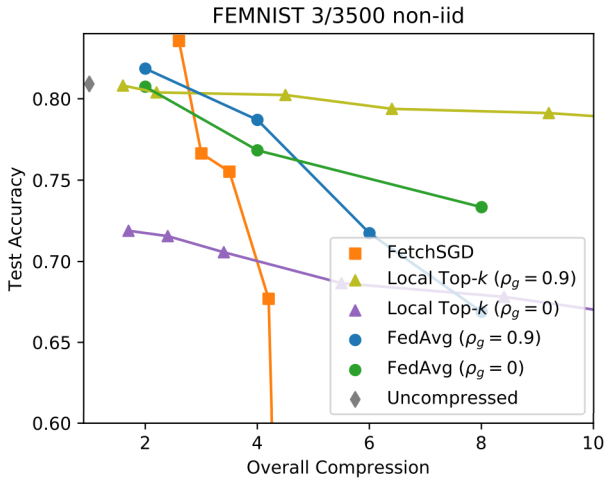
(b) PersonaChat Upload Compression



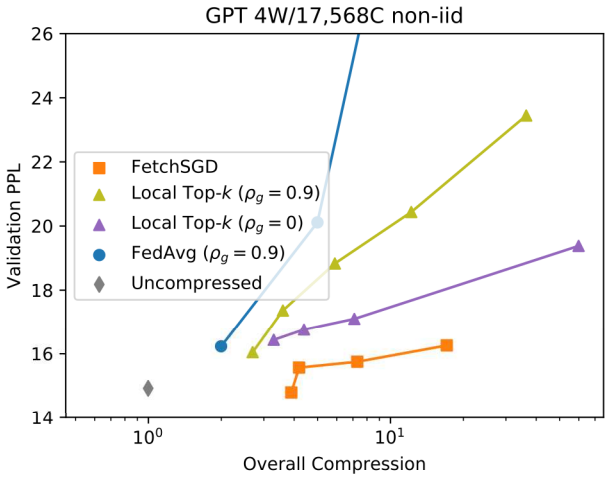
(c) FEMNIST Download Compression



(d) PersonaChat Download Compression

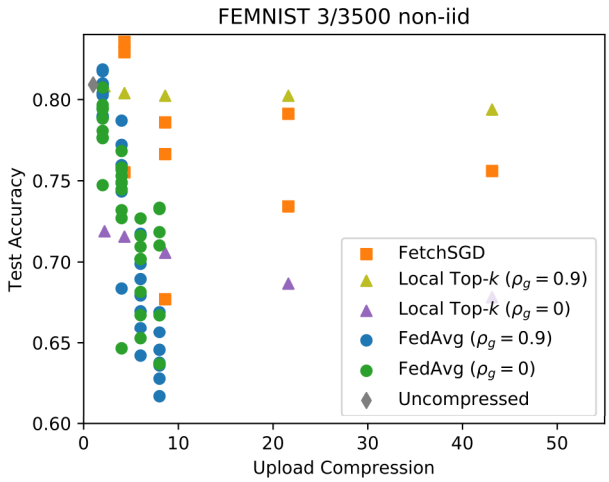


(e) FEMNIST Overall Compression

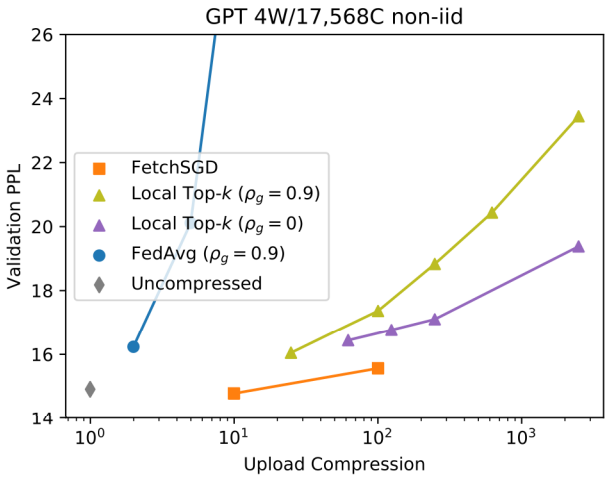


(f) PersonaChat Overall Compression

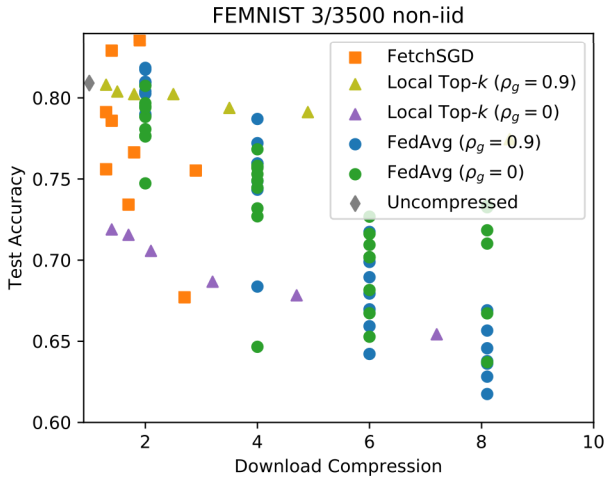
Figure 8. Upload (top), download (middle), and overall (bottom) compression for FEMNIST (left) and PersonaChat (right). To increase readability, each plot shows only the Pareto frontier of runs for the compression type shown in that plot. All results are shown in Figure 9.



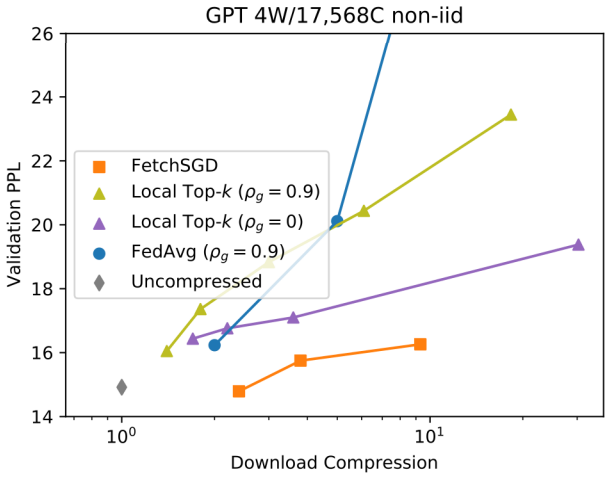
(a) FEMNIST Upload Compression



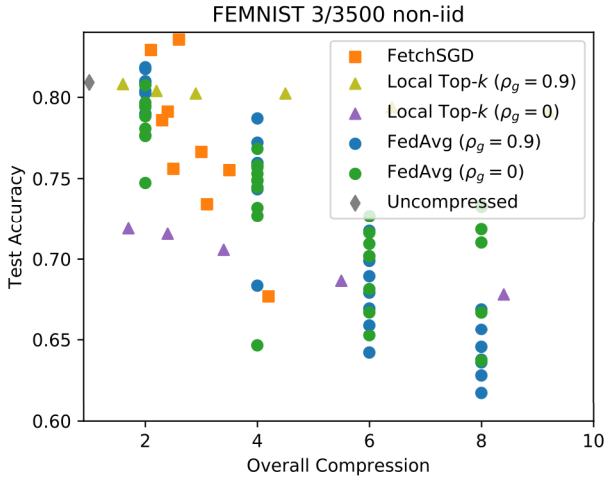
(b) PersonaChat Upload Compression



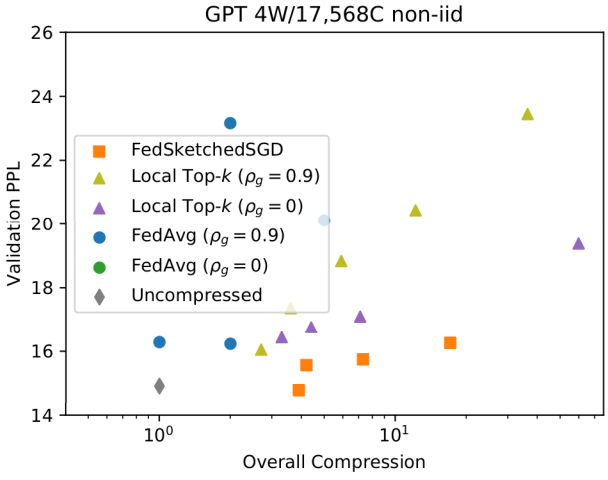
(c) FEMNIST Download Compression



(d) PersonaChat Download Compression



(e) FEMNIST Overall Compression



(f) PersonaChat Overall Compression

Figure 9. Upload (top), download (middle), and overall (bottom) compression for FEMNIST (left) and PersonaChat (right).

B. Theoretical properties

B.1. Proof of Theorem 1

We first verify that the stochastic gradients constructed are stochastic gradients with respect to the empirical mixture, and we calculate its second moment bound. At a given iterate w, we sample W clients *uniformly* from C clients at every iteration, and compute $\mathrm{g} = \frac{1}{W}\sum_{i=1}^{W} \mathrm{g}_i$, where g_i are stochastic gradients with respect to the distribution $\mathcal{D}_i$ on client i. This stochastic gradient is unbiased, as shown below.

$$\mathbb{E}\mathrm{g} = \widehat{\mathbb{E}}\mathbb{E}[\mathrm{g}|i] = \frac{1}{W}\frac{1}{\binom{C}{W}}\binom{C-1}{W-1}\sum_{i=1}^{C}\mathop{\mathbb{E}}_{\mathcal{D}_i}\mathrm{g}_i = \frac{1}{C}\sum_{i=1}^{C}\nabla f_i(\mathrm{w})$$

The norm of the stochastic gradient is bounded:

$$\mathbb{E}\|\mathrm{g}\|^2 = \widehat{\mathbb{E}}\mathbb{E}\left\|\frac{1}{W}\sum_{i\in B,|B|=W}\mathrm{g}_i \mid B\right\|^2 \leq \frac{1}{\binom{C}{W}}\frac{1}{W^2}W\binom{C-1}{W-1}\sum_{i=1}^{C}\mathop{\mathbb{E}}_{\mathcal{D}_i}\|\mathrm{g}_i\|^2 \leq \frac{\sum_{i=1}^{C}G_i^2}{C} =: G^2$$

This proof follows the analysis of compressed SGD with error feedback in Karimireddy et al. (2019b), with additional momentum. Let $C(\mathrm{x}) = \text{top-k}(U(S(\mathrm{x})))$, the error accumulation then is $S(\mathrm{e}_{t+1}) = S(\eta(\rho\mathrm{u}_{t-1} + \mathrm{g}_t) + \mathrm{e}_t) - S(C(\eta(\rho\mathrm{u}_{t-1} + \mathrm{g}_t) + \mathrm{e}_t))$. Consider the virtual sequence $\tilde{\mathrm{w}}_t = \mathrm{w}_t - \mathrm{e}_t - \frac{\eta\rho}{1-\rho}\mathrm{u}_{t-1}$. Upon expanding, we get

$$\begin{aligned}
\tilde{\mathrm{w}}_t &= \mathrm{w}_{t-1} - C(\eta(\rho\mathrm{u}_{t-2} + \mathrm{g}_{t-1}) + \mathrm{e}_{t-1}) + C(\eta(\rho\mathrm{u}_{t-2} + \mathrm{g}_{t-1}) + \mathrm{e}_{t-1}) - \eta(\rho\mathrm{u}_{t-2} + \mathrm{g}_{t-1}) - \mathrm{e}_{t-1} - \frac{\eta\rho}{1-\rho}\mathrm{u}_{t-1} \\
&= \mathrm{w}_{t-1} - \mathrm{e}_{t-1} - \eta\mathrm{g}_{t-1} - \eta\rho\mathrm{u}_{t-2} - \frac{\eta\rho}{1-\rho}(\rho\mathrm{u}_{t-2} + \mathrm{g}_{t-1}) \\
&= \mathrm{w}_{t-1} - \mathrm{e}_{t-1} - \eta\left(1 + \frac{\rho}{1-\rho}\right)\mathrm{g}_{t-1} - \eta\rho\left(1 + \frac{\rho}{1-\rho}\right)\mathrm{u}_{t-2} \\
&= \mathrm{w}_{t-1} - \mathrm{e}_{t-1} - \frac{\eta\rho}{1-\rho}\mathrm{u}_{t-2} - \frac{\eta}{1-\rho}\mathrm{g}_{t-1} \\
&= \tilde{\mathrm{w}}_{t-1} - \frac{\eta}{1-\rho}\mathrm{g}_{t-1}
\end{aligned}$$

So this reduces to an SGD-like update but with a scaled learning rate. Applying L-smoothness of f, we get,

$$\begin{aligned}
\mathbb{E}f(\tilde{\mathrm{w}}_{t+1}) &\leq f(\tilde{\mathrm{w}}_t) + \langle\nabla f(\tilde{\mathrm{w}}_t), \tilde{\mathrm{w}}_{t+1} - \tilde{\mathrm{w}}_t\rangle + \frac{L}{2}\|\tilde{\mathrm{w}}_{t+1} - \tilde{\mathrm{w}}_t\|^2 \\
&\leq f(\tilde{\mathrm{w}}_t) - \frac{\eta}{(1-\rho)}\mathbb{E}\langle\nabla f(\tilde{\mathrm{w}}_t), \mathrm{g}_t\rangle + \frac{L\eta^2}{2(1-\rho)^2}\mathbb{E}\|\mathrm{g}_t\|^2 \\
&\leq f(\tilde{\mathrm{w}}_t) - \frac{\eta}{(1-\rho)}\langle\nabla f(\tilde{\mathrm{w}}_t), \nabla f(\mathrm{w}_t)\rangle + \frac{L\eta^2}{2(1-\rho)^2}\mathbb{E}\|\mathrm{g}_t\|^2 \\
&\leq f(\tilde{\mathrm{w}}_t) - \frac{\eta}{(1-\rho)}\|\nabla f(\mathrm{w}_t)\|^2 + \frac{\eta}{2(1-\rho)}\left(\|\nabla f(\mathrm{w}_t)\|^2 + \mathbb{E}\|\nabla f(\tilde{\mathrm{w}}_t) - \nabla f(\mathrm{w}_t)\|^2\right) + \frac{L\eta^2G^2}{2(1-\rho)^2} \\
&\leq f(\tilde{\mathrm{w}}_t) - \frac{\eta}{2(1-\rho)}\|\nabla f(\mathrm{w}_t)\|^2 + \frac{\eta L^2}{2(1-\rho)}\mathbb{E}\|\tilde{\mathrm{w}}_t - \mathrm{w}_t\|^2 + L\eta^2\sigma^2 \\
&= f(\tilde{\mathrm{w}}_t) - \frac{\eta}{2(1-\rho)}\|\nabla f(\mathrm{w}_t)\|^2 + \frac{\eta L^2}{2(1-\rho)}\mathbb{E}\left\|\mathrm{e}_t + \frac{\eta\rho}{1-\rho}\mathrm{u}_{t-1}\right\|^2 + L\eta^2\sigma^2
\end{aligned}$$

We now need to bound $\left\|\mathrm{e}_t + \frac{\eta\rho}{1-\rho}\mathrm{u}_{t-1}\right\|^2$. However, we never compute or store e_t or u_t, since the algorithm only maintains sketches of e_t and u_t. Instead, we will bound $\left\|S(\mathrm{e}_t) + \frac{\eta\rho}{1-\rho}S(\mathrm{u}_{t-1})\right\|^2$. This is sufficient because $(1-\epsilon)\|\mathrm{x}\| \leq \|S(\mathrm{x})\| \leq$

$(1+\epsilon)\|\mathrm{x}\|$, for a user-specified constant epsilon. Note that $\left\|S(\mathrm{e}_t + \frac{\eta\rho}{1-\rho}\mathrm{u}_{t-1})\right\|^2 \leq 2\left(\|S(\mathrm{e}_t)\|^2 + \left(\frac{\eta\rho}{1-\rho}\right)^2 \|S(\mathrm{u}_{t-1})\|\right)$. We bound $\|S(\mathrm{u}_{t-1})\|$ first:

$$\|S(\mathrm{u}_{t-1})\|^2 = \left\|\sum_{i=1}^{t-1}\rho^i S(\mathrm{g}_i)\right\|^2 \leq \left(\sum_{i=1}^{t-1}\rho^i \|S(\mathrm{g}_i)\|\right)^2 \leq \left(\frac{(1+\epsilon)G}{1-\rho}\right)^2$$

For the other term, we expand it and bound as

$$\begin{aligned}
\|S(\mathrm{e}_t)\|^2 &\leq (1-\tau)\|\eta(\rho S(\mathrm{u}_{t-1}) + S(\mathrm{g}_{t-1})) + S(\mathrm{e}_{t-1})\|^2 \\
&\leq (1-\tau)\left((1+\gamma)\|S(\mathrm{e}_{t-1})\|^2 + (1+1/\gamma)\eta^2\|S(\mathrm{u}_t)\|^2\right) \\
&\leq (1-\tau)\left((1+\gamma)\|S(\mathrm{e}_{t-1})\|^2 + \frac{(1+1/\gamma)(1+\epsilon)^2\eta^2G^2}{(1-\rho)^2}\right) \\
&\leq \sum_{i=0}^{\infty}\frac{(1+\epsilon)^2((1-\tau)(1+\gamma))^i(1+1/\gamma)\eta^2G^2}{(1-\rho^2)} \\
&\leq \frac{(1+\epsilon)^2(1-\tau)(1+1/\gamma)\eta^2G^2}{1-((1-\tau)(1+\gamma))}.
\end{aligned}$$

where in the second inequality, we use the inequality $(a+b)^2 \leq (1+\gamma)a^2 + (1+1/\gamma)b^2$. As argued in Karimireddy et al. (2019b), choosing $\gamma = \frac{\tau}{2(1-\tau)}$ suffices to upper bound the above with $\leq \frac{4(1+\epsilon)^2(1-\tau)\eta^2G^2}{\tau^2(1-\rho)^2}$.

Plugging everything in, choosing $\eta \leq (1-\rho)/2L$, and rearranging, we get that

$$\|\nabla f(\mathrm{w}_t)\|^2 \leq \frac{2(1-\rho)}{\eta}\left(f(\tilde{\mathrm{w}}_t) - \mathbb{E}f(\tilde{\mathrm{w}}_{t+1}) + \frac{\eta L^2}{2(1-\rho)}\frac{4(1+\epsilon)^2(1-\tau)\eta^2G^2}{(1-\epsilon)^2\tau^2(1-\rho)^2} + L\eta^2\sigma^2\right).$$

Averaging and taking expectations gives us

$$\min_{t=1\cdots T}\mathbb{E}\|\nabla f(\mathrm{w}_t)\|^2 \leq \frac{1}{T}\sum_{t=1}^{T}\mathbb{E}\|\nabla f(\mathrm{w}_t)\|^2 \leq \frac{2(1-\rho)(f(\mathrm{w}_0)-f^*)}{\eta T} + \frac{4L^2(1+\epsilon)^2(1-\delta)\eta^2G^2}{(1-\epsilon)^2\delta^2(1-\rho)^2} + 2L(1-\rho)\eta\sigma^2.$$

Finally, choosing $\epsilon = 1/k$ and setting $\eta = \frac{1-\rho}{2L\sqrt{T}}$ finishes the proof.

Also, note that setting the momentum $\rho = 0$ in the above, we recover a guarantee for `FetchSGD` with no momentum

Corollary 1. *For a L-smooth non-convex function f, `FetchSGD`, with no momentum, under Assumption 1, with stochastic gradients of norm bounded by G and variance bounded by σ^2, in T iterations, returns $\hat{\mathrm{w}}_T$ such that*

$$\min_{t=1\cdots T}\mathbb{E}\|f(\mathrm{w}_t)\|^2 \leq \frac{4L(f(\mathrm{w}_0)-f^*)+\sigma^2}{\sqrt{T}} + \frac{(1+\epsilon)^2(1-\tau)G^2}{2(1-\epsilon)^2\delta^2T}$$

B.2. Proof of Theorem 2

B.2.1. Warm-up: $I = 1$ (without error accumulation)

Let us first consider a simple case where we only use the heavy hitters in the current gradient with no error accumulation. The update is of the form

$$\mathrm{w}_{t+1} = \mathrm{w}_t - C(\eta\mathrm{g}_t)$$

Note that C here is FindHeavy$_\alpha$. Consider the virtual sequence $\tilde{\mathrm{w}}_t = \mathrm{w}_t - \sum_{i=1}^{t-1}(\eta\mathrm{g}_i - C(\eta\mathrm{g}_i))$. Upon expanding, we get

$$\tilde{\mathrm{w}}_t = \mathrm{w}_{t-1} - C(\eta\mathrm{g}_{t-1}) - \sum_{i=1}^{t-1}(\eta\mathrm{g}_i - C(\eta\mathrm{g}_i)) = \mathrm{w}_{t-1} - \sum_{i=1}^{t-2}(\eta\mathrm{g}_i - C(\eta\mathrm{g}_i)) - \eta\mathrm{g}_{t-1} = \tilde{\mathrm{w}}_{t-1} - \eta\mathrm{g}_{t-1}$$

Therefore $\tilde{w}_t - w_t = -\sum_{i=1}^{t-1}(\eta g_i - C(\eta g_i))$. In the following analysis we will see that we need to control $\|\tilde{w}_t - w_t\|$. From L-smoothness of f,

$$\begin{aligned}
\mathbb{E}f(\tilde{w}_{t+1}) &\leq f(\tilde{w}_t) + \mathbb{E}\langle \nabla f(\tilde{w}_t), \tilde{w}_{t+1} - \tilde{w}_t\rangle + \frac{L}{2}\mathbb{E}\|\tilde{w}_{t+1} - \tilde{w}_t\|^2 \\
&= f(\tilde{w}_t) - \mathbb{E}\eta\,\langle \nabla f(\tilde{w}_t), g_t\rangle + \frac{L\eta^2}{2}\mathbb{E}\|g_t\|^2 \\
&\leq f(\tilde{w}_t) - \eta\,\langle \nabla f(\tilde{w}_t) - \nabla f(w_t) + \nabla f(w_t), \nabla f(w_t)\rangle + \frac{LG^2\eta^2}{2} \\
&\leq f(\tilde{w}_t) - \eta\,\|\nabla f(w_t)\|^2 + \frac{\eta}{2}\left(\|\nabla f(w_t)\|^2 + \|\nabla f(\tilde{w}_t) - \nabla f(w_t)\|^2\right) + \frac{\eta^2 LG^2}{2} \\
&\leq f(\tilde{w}_t) - \frac{\eta}{2}\,\|\nabla f(w_t)\|^2 + \frac{\eta L}{2}\mathbb{E}\|\tilde{w}_t - w_t\|^2 + \frac{\eta^2 LG^2}{2}
\end{aligned}$$

where in the third inequality, we used $\langle u, v\rangle \leq \frac{1}{2}\left(\|u\|^2 + \|v\|^2\right)$. Note we need to bound $\|\tilde{w}_t - w_t\| = \left\|\sum_{i=1}^{t-1}(C(\eta g_i) - \eta g_i)\right\|$. Our compression operator $C(x) = \text{FindHeavy}_\alpha(U(S(x)))$ i.e. it recovers all α heavy coordinates from x. Every $(1,\alpha)$ sliding heavy sequence of gradients by assumption contains at least one α-heavy hitter in every gradient with probability $1-\delta$ and our compression operator recovers all of them.Therefore, by Assumption 1, $x - C(x)$ only has the heavy-hitter estimation error plus non-heavy noise. Therefore, conditioned on the heavy hitter recovery event, $\tilde{w}_t - w$ is estimation error + noise. The first term – the estimation error – is the sketch's heavy-hitter estimation error, and it has mean zero and is symmetric. This is because the the count sketch produces unbiased estimates of coordinates, and because it uses uniformly random hash functions, the probability to output a value of either side of the mean is equal. The second term – the noise – also has mean zero and is symmetric, by Assumption 1. Formally, the estimation error and noise are of the form $z_i = \|g_i\|\,\xi_i$, where the ξ's are mutually independent and independent of $\|g_i\|$. Hence, z_i is symmetric noise of a constant (independent) scale relative to the gradient size. It is therefore the case that

$$\|\tilde{w}_t - w_t\| = \left\|\sum_{i=1}^{t-1}(C(\eta g_i) - \eta g_i)\right\| = \eta\left\|\sum_{i=1}^{t-1}(\text{estimation error}_i) + (\text{noise}_i)\right\| = \eta\left\|\sum_{i=1}^{t-1} z_i\right\|.$$

Note that since the g_i's are dependent because they are a sequence of SGD updates, the z_i's are also dependent. However since the ξ_i's are independent with mean zero, $\mathbb{E}[\|g_i\|\,\xi_i|\mathcal{F}_i] = 0$, where $\mathcal{F}_i$ is the filtration of events before the t^{th} iteration. So the stochastic process $\{\|g_i\|\,\xi_i\}_{i=1}^t$ forms a martingale difference sequence. For a martingale difference sequence $\{x_i\}_{i=1}^T$, it holds that

$$\mathbb{E}\left\|\sum_{i=1}^{T} x_i - \mathbb{E}x_i\right\|^2 = \sum_{i=1}^{T}\mathbb{E}\|x_i - \mathbb{E}x_i\|^2 + \sum_{i,j=1,i\neq j}^{T,T}\mathbb{E}\langle x_i - \mathbb{E}x_i, x_j - \mathbb{E}x_j\rangle.$$

For $i > j$, $\mathbb{E}\langle x_i - \mathbb{E}x_i, x_j - \mathbb{E}x_j\rangle = \mathbb{E}_j\mathbb{E}\langle x_i - \mathbb{E}x_i, x_j - \mathbb{E}x_j\rangle|j = 0$. Applying this, we get

$$\mathbb{E}\left\|\sum_{i=1}^{t-1}(C(\eta g_i) - \eta g_i)\right\|^2 = \eta^2\mathbb{E}\left\|\sum_{i=1}^{t-1}\|g_i\|\,\xi_i\right\|^2 = \eta^2\sum_{i=1}^{t-1}\mathbb{E}\|\|g_i\|\,\xi_i\|^2 = \eta^2\sum_{i=1}^{t-1}\mathbb{E}\|z_i\|^2,$$

where in the last equality, we applied the martingale difference result noting that the random variable $z_i = \|g_i\|\,\xi_i$ are mean zero. We will now look how heavy coordinates and noise coordinates contribute to the norm of z_i. Let $z_i^{\text{estimation}}$ and z_i^{noise} be the estimation error vector and noise vector, respectively. Potentially all coordinates have noise, and some of these also have estimation error, so we can decompose $\|z_i\|^2 = \|z_i^{\text{estimation}} + z_i^{\text{noise}}\|^2 \leq 2(\|z_i^{\text{estimation}}\|^2 + \|z_i^{\text{noise}}\|^2)$.

From Lemma 2 in (Charikar et al., 2002), for each bucket in the Count Sketch, the variance in estimation is at most the ℓ_2 norm of the tail divided by the number of buckets b. Since the tail has mass at most $(1-\alpha)G^2$, for each coordinate j, we

(a) Naive sliding window

(b) Adaptive sliding window

Figure 11. Approximating ℓ_2 norm on sliding windows.

have $\mathbb{E}((\mathrm{z}_i^{\mathrm{H}})_j)^2 \leq \frac{(1-\alpha)G^2}{b}$. There is at most $1/\alpha^2$ heavy coordinates present, and the number of buckets b is chosen to be larger than $1/\alpha^2$. Thus $\mathbb{E}\left\|\mathrm{z}_i^{\text{estimation}}\right\|^2 \leq \frac{1}{\alpha^2} \cdot \frac{(1-\alpha)G^2}{b} \leq (1-\alpha)G^2$. Also, $\mathbb{E}\left\|\mathrm{z}_i^{\text{noise}}\right\|^2 \leq \beta G^2$, since the noise only occupies a β fraction of the the gradient, so its norm is at most βG^2.

Plugging this in, taking $\|\nabla f(\mathrm{w}_t)\|$ to the left hand side, averaging and taking expectation with respect to all randomness, we get that

$$\min_{t=1\cdots T} \mathbb{E}\|\nabla f(\mathrm{w}_t)\|^2 \leq \frac{\sum_{i=1}^T \mathbb{E}\|\nabla f(\mathrm{w}_i)\|^2}{T} \leq \frac{(f(\mathrm{w}_0) - f^*)}{\eta T} + \eta^2 2(1-\alpha+\beta)G^2 LT + L\eta G^2$$

Finally choosing $\eta = \frac{1}{G\sqrt{L}T^{2/3}}$, we get,

$$\min_{t=1\cdots T} \mathbb{E}\|\nabla f(\mathrm{w}_t)\|^2 \leq \frac{G\sqrt{L}\left((f(\mathrm{w}_0) - f^*) + 2(1-\alpha+\beta)\right))}{T^{1/3}} + \frac{G\sqrt{L}}{T^{2/3}}$$

Note that the analysis above holds conditioned on the success of the Count Sketch data structure and on the event that the first statement in Definition 1 holds at every iteration, which happens with probability $1 - 2T\delta$ by union bound. This leads to the size of the sketch provided in Theorem 2.

B.2.2. General case: any I (with error accumulation)

Intuition. We first give some motivation for why the (I, α) sliding heavy assumption helps. Definition 1 specifies the structure of the noise included in every gradient, s.t. the growth of the norm of the noise is bounded by $O(\sqrt{t})$. Intuitively, discarding the noise during the training does not greatly hurt the convergence. However noise does hurt the recovery of heavy coordinates. A Count Sketch finds heavy coordinates in the entire input, which is $\eta \mathrm{g}_t + \mathrm{e}_t$, but coordinates are heavy only on an interval of length at most I. For example, half way to convergence, $\|\mathrm{e}_t\|$ will grow up to $\mathcal{O}\left(\sqrt{T/2}\right)$, while coordinates heavy on an interval of size I are bounded by $\mathcal{O}(I) \ll \sqrt{T/2}$. To avoid this problem, we suggest regularly cleaning up the error accumulation: error collected on steps $1, \ldots, t-I$ can not contain the signal that is not recovered at the moment t. However, a similar argument shows that maintaining e_t over the last I updates is not sufficient due to variability in how wide of an interval the heavy coordinate can be spread over. Therefore we suggest maintaining I windows $\left\{\mathrm{e}_t^i\right\}_{i=1}^T$ of sizes $\{1, 2, \ldots, I\}$, cleaning up each error accumulation sketch every I updates correspondingly (see Figure 11a). In this configuration, at any moment t and for any I' (see Definition 1), there exist j s.t. error accumulation e_t^j was cleaned up at time moment $t - I'$, and that Count Sketch can detect the signal from $\mathrm{g}_{t,1}$. This approach finds heavy coordinates in any suffix of updates $(\mathrm{g}_{t-I'}, \ldots, \mathrm{g}_t)$ for $I' < I$. Recall that we maintain accumulated error inside sketches, thus maintaining I windows requires I sketches. Though sketch operations are very efficient and its memory size is sub-linear, linear dependency on I is unfavorable and limits the choice of I. Finding heavy coordinates in sliding windows of streaming data is a well studied area with several algorithms to find heavy coordinates (Braverman and Ostrovsky, 2007; Datar et al., 2002). In Appendix D we briefly discuss sliding window techniques and how they can help to reduce the number of sketches to $\log(I)$.

Proof. In the general case, we have an error accumulation data structure S_t, which takes the sketches of the the new gradient $S(\mathrm{g}_t)$. The data structure is essentially a function of the last I gradients, and has a function FindHeavy, which

returns a vector C_t which *captures* the heavy coordinates in the sum of the last I' gradients, for any $I' < I$. We use this to update the model, and after this, the data structure is updated to *forget* C_t as well as anything other than the last I gradients. For the sake of clarity, we first assume there exists such a data structure. We will later describe how to use count sketches to create such a data structure as well as discuss an efficient implementation. The procedure is formalized below.

$$\begin{aligned}
S_{t+1} &= \text{Insert}(S_t, \eta S(\mathrm{g}_t)) \\
C_t &= \text{FindHeavy}(S_{t+1}) \\
\mathrm{w}_{t+1} &= \mathrm{w}_t - C_t \\
S_{t+1} &= \text{Update}(S_{t+1}, C_t)
\end{aligned}$$

As in the warm-up case, consider a virtual sequence

$$\begin{aligned}
\tilde{\mathrm{w}}_t &= \mathrm{w}_t - (\sum_{i=1}^{t-1} \eta \mathrm{g}_i - C_i) \\
&= \mathrm{w}_{t-1} - C_{t-1} - (\sum_{i=1}^{t-1} \eta \mathrm{g}_i - C_i) \\
&= \mathrm{w}_{t-1} - \sum_{i=1}^{t-2} \eta \mathrm{g}_i - C_i - \eta \mathrm{g}_{t-1} \\
&= \tilde{\mathrm{w}}_{t-1} - \eta \mathrm{g}_{t-1}
\end{aligned}$$

We have $\tilde{\mathrm{w}}_t - \mathrm{w}_t = \sum_{i=1}^{t-1} \eta \mathrm{g}_i - C_i$.

$$\tilde{\mathrm{w}}_t - \mathrm{w}_t = \sum_{i=1}^{t-1} \eta \mathrm{g}_i^N + \sum_{i=1}^{t-1} \eta \mathrm{g}_i^S - \sum_{i=1}^{t-1} C_i$$

where g_i^N is the noise and g_i^S represents signal. $\forall i < t - I$, the sliding window Count Sketch data structure will recover all the signal, and for $t - I \leq i \leq t$, some signal remains, to be recovered in future steps. Since the gradients are bounded in norm, the norm of the sum of the past I gradients, from which signal has yet to be recovered, can be bounded as IG. As shown in the warm-up case, we argue that

$$\begin{aligned}
\tilde{\mathrm{w}}_t - \mathrm{w}_t &= \sum_{i=1}^{t} (\eta \mathrm{g}_i - \mathrm{C}_i) = \sum_{i=1}^{t-I} \eta \mathrm{g}_i - \sum_{i=1}^{t} C_i + \sum_{i=t-I+1}^{t} \eta \mathrm{g}_i \\
&= \sum_{i=1}^{t-I} \text{estimation error}_i + \text{noise}_i + \sum_{i=t-I+1}^{t} \eta \mathrm{g}_i = \sum_{i=1}^{t-1} \mathrm{z}_i + \sum_{i=t-I+1}^{t} \eta \mathrm{g}_i
\end{aligned}$$

This follows because the sliding window data structure recovers all the signal in the sum of last I gradients. Then, by the triangle inequality we get

$$\|\tilde{\mathrm{w}}_t - \mathrm{w}_t\|^2 \leq 2 \left\| \sum_{i=1}^{t-1} \mathrm{z}_i \right\|^2 + 2\eta^2 I^2 G^2$$

We now similarly argue that z_i forms a martingale difference sequence and therefore we have

$$\mathbb{E}\|\tilde{\mathrm{w}}_t - \mathrm{w}_t\|^2 \leq 2\mathbb{E}\| \sum_{i=1}^{t} \mathrm{z}_i \| + 2\eta^2 I^2 G^2 \leq 2(1 - \alpha + \beta)\eta^2 G^2 + 2\eta^2 I^2 G^2$$

Repeating the steps in the warm-up case: using L-smoothness of f, we get

$$\begin{aligned}\mathbb{E}f(\tilde{\mathrm{w}}_{t+1}) &\leq f(\tilde{\mathrm{w}}_t) - \frac{\eta}{2}\left\|\nabla f(\mathrm{w}_t)\right\|^2 + \frac{\eta L}{2}\mathbb{E}\|\tilde{\mathrm{w}}_t - \mathrm{w}_t\|^2 + \frac{\eta^2 L G^2}{2} \\ &\leq f(\tilde{\mathrm{w}}_t) - \frac{\eta}{2}\left\|\nabla f(\mathrm{w}_t)\right\|^2 + \frac{\eta L}{2}\left(2(1-\alpha+\beta)\eta^2 G^2 + 2\eta^2 I^2 G^2\right) + \frac{\eta^2 L G^2}{2}\end{aligned}$$

Taking $\|\nabla f(\mathrm{w}_t)\|$ to the left hand side, averaging and taking expectation with respect to all randomness, and choosing $\eta = \frac{1}{G\sqrt{L}T^{2/3}}$ we get

$$\min_{t=1\cdots T}\mathbb{E}\|\nabla f(\mathrm{w}_t)\|^2 \leq \frac{G\sqrt{L}\left(\left(f(\mathrm{w}_0) - f^*\right) + 2(1-\alpha+\beta) + 2I^2\right)}{T^{1/3}} + \frac{G\sqrt{L}}{T^{2/3}}$$

The first part of the theorem is recovered by noting that $\beta \leq 1$. For the second part, note that the size of sketch needed to capture α-heavy hitters with probability at least $1-\delta$ is $\mathcal{O}\left(\frac{\log(d\delta)}{\alpha^2}\right)$; taking a union bound over all T iterations recovers the second claim in the theorem.

Implementation. We now give details on how this data structure is constructed and what the operations correspond to. For all heavy coordinates to be successfully recovered from all suffixes of the last I gradient updates (i.e. $\forall I' < I$, to recover heavy coordinates of $\sum_{i=t-I'}^{t}\eta \mathrm{g}_i$) we can maintain I sketches in the overlapping manner depicted in Figure 11a. That is, every sketch is cleared every I iterations. To find heavy coordinates, the FindHeavy() method must query every sketch and return the united set of heavy coordinates found; Insert() appends new gradients to all I sketches; and Update() subtracts the input set of heavy coordinates from all I sketches. Although sketches are computationally efficient and use memory sub-linear in d (a Count Sketch stores $\mathcal{O}\left(\log\left(d\right)\right)$ entries), linear dependency on I in unfavorable, as it limits our choice of I. Fortunately, the sliding window model, which is very close to the setting studied here, is thoroughly studied in the streaming community (Braverman and Ostrovsky, 2007; Datar et al., 2002). These methods allow us to maintain a number of sketches only logarithmic in I. For a high level overview we refer the reader to Appendix D.

Are these assumptions necessary? We have discussed that un-sketching a sketch gives an unbiased estimate of the gradient: $\mathbb{E}\mathcal{U}(\mathcal{S}(\mathrm{g})) = \mathrm{g}$, so the sketch can be viewed as a stochastic gradient estimate. Moreover, since Top-k, error feedback and momentum operate on these new stochastic gradients, existing analysis can show that our method converges. However, the variance of the estimate derived from unsketching is $\Theta(d)$, in the worst-case. By standard SGD analysis, this gives a convergence rate of $\mathcal{O}\left(d/\sqrt{T}\right)$, which is optimal since the model is a function of only these new $\mathcal{O}\left(d\right)$-variance stochastic gradients. This establishes that even without any assumptions on the sequence of gradients encountered during optimization, our algorithm has convergence properties. However this dimensionality dependence does not reflect our observation that the algorithm performs competitively with uncompressed SGD in practice, motivating our assumptions and analysis.

C. Count Sketch

Streaming algorithms have aided the handling of enormous data flows for more than two decades. These algorithms operate on sequential data updates, and their memory consumption is sub-linear in the problem size (length of stream and universe size). First formalized in (Alon et al., 1999), sketching (a term often used for streaming data structures) facilitates numerous applications, from handling networking traffic (Ivkin et al., 2019b) to analyzing cosmology simulations (Liu et al., 2015). In this section we provide a high-level overview of the streaming model, and we explain the intuition behind the Count Sketch (Charikar et al., 2002) data structure, which we use in our main result. For more details on the field, we refer readers to (Muthukrishnan et al., 2005).

Consider a frequency vector $g \in \mathrm{R}^d$ initialized with zeros and updated coordinate by coordinate in the streaming fashion – i.e. at time t update (a_i, w_i) changes the frequency as $g_{a_i} += w_i$. Alon et al. (1999) introduces the AMS sketch, which can approximate $\|g\|$ with only constant memory. Memory footprint is very important in streaming settings, as d is usually

assumed to be too large for g to fit into the memory. The AMS sketch consists of a running sum S initialized with 0, and a hash function h that maps coordinates of g into ± 1 in an i.i.d. manner. Upon arrival of an update (a_i, w_i), the AMS sketch performs a running sum update: $S \mathrel{+}= h(a_i)w_i$. Note that at the end of the stream, $\mathbb{E}(S) = \sum_{i=1}^{n} h(a_i)w_i$ can be reorganized as per coordinate $\mathbb{E}(S) = \sum_{j=1}^{d} \left(h(j) \sum_{\{i:a_i=j\}} w_i \right) = \sum_{j=1}^{d} h(j)g_j$, where g_j is the value of j-th coordinate at the end of the stream. The AMS sketch returns S^2 as an estimation of $\|g\|^2$: $\mathbb{E}(S^2) = \mathbb{E}(\sum_{j=1}^{d} h(j)^2 g_j^2) + \mathbb{E}(\sum_{j=1}^{d} h(j)h(j')g_j g_{j'})$. If h is at least 2-wise independent second, then both $\mathbb{E}h(j)h(j')$ and the second term are 0. So $\mathbb{E}(S^2) = \mathbb{E}(\sum_{j=1}^{d} g_j^2) = \|g\|^2$, as desired. Similarly, Alon et al. (1999) show how to bound the variance of the estimator (at the cost of 4-wise hash independence). The AMS sketch maintains a group of basic sketches described above, so that the variance and failure probability can be controlled directly via the amount of memory allocated: an AMS sketch finds $\widehat{\ell}_2 = \|g\| \pm \varepsilon\|g\|$ using $O(1/\varepsilon^2)$ memory.

The Count Sketch data structure (Charikar et al., 2002) extends this technique to find heavy coordinates of the vector. A coordinate i is (α, ℓ_2)-heavy (or an (α, ℓ_2)-heavy hitter) if $g_i \geq \alpha\|g\|$. The intuition behind the Count Sketch is as follows: the data structure maintains a hash table of size c, where every coordinate $j \in [d]$ is mapped to one of the bins, in which an AMS-style running sum is maintained. By definition, the heavy coordinates encompass a large portion of the ℓ_2 mass, so the ℓ_2 norm of the bins where heavy coordinates are mapped to will be significantly larger then that of the rest of the bins. Consequently, coordinates mapped to the bins with small ℓ_2 norm are not heavy, and can be excluded from list of heavy candidates. Repeating the procedure $O(\log(d))$ times in parallel reveals the identities of heavy coordinates and estimates their values. Formally, a Count Sketch finds all (α, ℓ_2)-heavy coordinates and approximates their values with $\pm\varepsilon\|g\|$ additive error. It requires $O(\frac{1}{\varepsilon^2\alpha^2}\log(d))$ memory. Algorithm 2 depicts the most important steps in a Count Sketch. For more details on the proof and implementation, refer to (Charikar et al., 2002).

Algorithm 2 Count Sketch (Charikar et al., 2002)

function init(r, c):
  init $r \times c$ table of counters S
  for each row r init sign and bucket hashes: $\left\{(h_j^s, h_j^b)\right\}_{j=1}^{r}$
function update((a_i, w_i)):
  for j in $1 \ldots r$: $\quad S[j, h_j^b(i)] \mathrel{+}= h_j^s(i)w_i$
function estimate(i):
  init length r array estimates
  for j in $1, \ldots, r$:
    estimates$[r] = h_j^s(i)S[j, h_j^b(i)]$
  return median(estimates)

For `FetchSGD`, an important feature of the Count Sketch data structure is that it is linear – *i.e.*, $S(g_1) + S(g_2) = S(g_1 + g_2)$. This property is used when combining the sketches of gradients computed on every iteration, and to maintain error accumulation and momentum. We emphasize that while there are more efficient algorithms for finding heavy hitters, they either provide weaker ℓ_1 approximation guarantees (Muthukrishnan et al., 2005) or support only non-negative entries of the vector (Misra and Gries, 1982; Braverman et al., 2017). The structure of the Count Sketch allows for high amounts of parallelization, and the operations of a Count Sketch can be easily accelerated using GPUs (Ivkin et al., 2018).

D. Sliding Windows

As was mentioned in Appendix C, the streaming model focuses on problems where data items arrive sequentially and their volume is too large to store on disk. In this case, accessing previous updates is prohibited, unless they are stored in the sketch. In many cases, the stream is assumed to be infinite and the ultimate goal is to approximate some function on the last n updates and to *"forget"* the older ones. The sliding window model, introduced in (Datar et al., 2002), addresses exactly this setting. Recall the example from Appendix C: given a stream of updates (a_t, w_t) to a frequency vector g (i.e. $(g_t)_{a_t} \mathrel{+}= w_t$), approximating the ℓ_2 norm of g in the streaming model implies finding $\widehat{\ell}_2 = \|g\| \pm \varepsilon\|g\|$ On the other hand, in the sliding window model one is interested only in the last n updates, i.e. $\widehat{\ell}_2 = \|g_t - g_{t-n}\| \pm \varepsilon\|g_t - g_{t-n}\|$.

One naive solution is to maintain n overlapping sketches, as in Fig. 11a. However, such a solution is infeasible for larger n. Currently there are 2 major frameworks to *adopt* streaming sketches to the sliding window model: exponential histograms,

by Datar et al. (2002), and smooth histograms, by Braverman and Ostrovsky (2007). For simplicity, we will provide only the high level intuition behind the latter one. Maintaining all n sketches as in Fig. 11a is unnecessary if one can control the growth of the function: neighboring sketches differ only by one gradient update, and the majority of the sketches can be pruned. Braverman and Ostrovsky (2007) show that if a function is monotonic and satisfies a smoothness property, then the sketches can be efficiently pruned, leaving only $\mathcal{O}(\log(n))$ sketches. As in Fig. 11b, $\|S^{(i)}\| < (1+\varepsilon)\|S^{(i-1)}\|$, so any value in the intermediate suffixes (which were pruned earlier) can be approximated by the closest sketch $\|S^{(i)}\|$. For more details on how to construct this data structure, and for a definition of the smoothness property, we refer readers to Braverman and Ostrovsky (2007).