Communication-Efficient Federated Learning with Sketching

Ashwinee Panda
Daniel Rothchild
Enayat Ullah
Nikita Ivkin
Ion Stoica
Joseph Gonzalez, Ed.
Raman Arora, Ed.
Vladimir Braverman, Ed.

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2020-57
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2020/EECS-2020-57.html

May 23, 2020

COMMUNICATION-EFFICIENT FEDERATED LEARNING WITH SKETCHING

A PREPRINT

Daniel Rothchild[1*] **Ashwinee Panda**[1*] **Enayat Ullah**[2] **Nikita Ivkin**[3†]

Vladimir Braverman[2] **Joseph Gonzalez**[1] **Ion Stoica**[1] **Raman Arora**[2]

[1] UC Berkeley, {drothchild, ashwineepanda, jegonzal, istoica}@berkeley.edu
[2]Johns Hopkins University, enayat@jhu.edu, {vova, arora}@cs.jhu.edu
[3] Amazon, ivkin@amazon.com

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 FedSketchedSGD, to overcome these challenges. FedSketchedSGD 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 FedSketchedSGD 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. We prove that FedSketchedSGD 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 (Nishio and Yonetani, 2019).

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 updates to the current model based on their local data using Stochastic Gradient Descent (SGD) (Robbins and Monro, 1951), 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., 2019c), 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 quality models under the constraints imposed by the federated setting.

There are three main constraints unique to the federated setting that make training 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,

*equal contribution
†Work was done while at Johns Hopkins University

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).

Existing methods for addressing these constraints can achieve high performance in certain settings (McMahan et al., 2016). In particular, these approaches do well when local client datasets are relatively large, and when the non-i.i.d. nature of the data is relatively mild. However, we find that, consistent with prior work, existing methods can degrade in performance when using more drastically non-i.i.d. data, and when clients have small local datasets (Sattler et al., 2019; Smith et al., 2017). The ability to train on small local datasets is particularly important, 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). Furthermore, if data accumulates slowly on the edge devices, waiting for many data points to accumulate may be undesirable, since doing so delays learning on the new data.

In this paper, we propose a new optimization algorithm for federated learning, called `FedSketchedSGD`, that can train large quality models on non-i.i.d. data under communication constraints. Our method (Fig. 1) requires no local state, and scales seamlessly to very small local datasets. In `FedSketchedSGD`, clients compress their gradients using a randomized data structure called a Count Sketch (Charikar et al., 2002) before transmitting them to the central aggregator. In addition, `FedSketchedSGD` can leverage momentum, a technique that is essential for attaining high accuracy in the non-federated setting (Sutskever et al., 2013). Note that previously there was no consensus on successfully incorporating momentum into existing algorithms for the federated setting (Lin et al., 2018). Lastly, we show that under mild assumptions, `FedSketchedSGD` has favourable convergence guarantee in the non-i.i.d. setting while also achieving communication efficiency.

We 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.

In summary, we make the following contributions:

1. We introduce `FedSketchedSGD`, an algorithm for training machine learning models in the federated setting without requiring any local state, and we give a convergence rate in the non-i.i.d. setting for L-smooth non-convex functions. The theoretical result is not optimization theoretic as it is obtained under assumptions, motivated from practice, on the sequence of updates (see Section 4) in the optimization trajectory.
2. We show that, without using local state, `FedSketchedSGD` can carry out momentum as if on local gradients before compression.
3. We demonstrate empirically that `FedSketchedSGD` outperforms two popular methods for increasing communication efficiency in federated learning, in the regime of non-i.i.d. data with small local datasets: top-k sparsification and `FedAvg`.

We give an overview of `FedSketchedSGD` in Fig. 1.

2 Related Work

Three main challenges make training high-quality models difficult in the federated setting: the need for communication efficiency, the inability to use local state on the clients, and the non-i.i.d. nature of federated data. Prior work mostly addresses only one or two of these challenges at a time. `FedSketchedSGD` is able to achieve communication efficiency without using local state when training on non-i.i.d. datasets, seamlessly handling all three challenges.

2.1 Communication Efficiency

Broadly speaking, there are two strategies that can be used to increase communication efficiency: 1) increase the amount of local computation to be done between communication rounds in such a way that the total number of communication rounds can be decreased, or 2) decrease the amount of data that is sent in each communication round, while not increasing the number of communication rounds significantly.

Federated learning can increase local computation by carrying out multiple steps of 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 local SGD, every participating client first downloads and trains the global model on their local dataset for a number of epochs using SGD (Stich, 2018). 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. Methods that use momentum in conjunction with local SGD have been proposed (Wang et al., 2019b) but the theoretical guarantees of using such methods in the federated setting are unclear.

`FedAvg` has convergence guarantees for the IID setting (Wang and Joshi, 2018). In order to achieve favorable compression, `FedAvg` has to do a significant amount of local computation in each iteration. When these local datasets become too small, as in our evaluation, we show

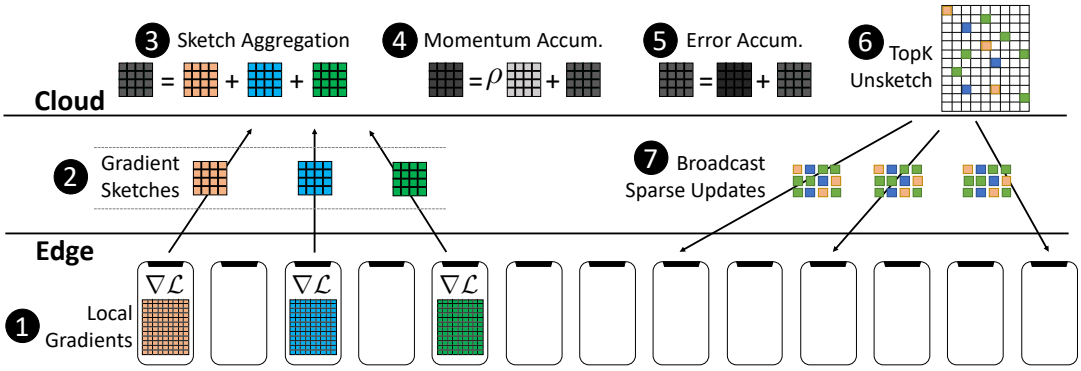


Figure 1: **Algorithm Overview**. The `FedSketchedSGD` algorithm **(1)** computes gradients locally, and then send sketches **(2)** of the gradients to the cloud. In the cloud, gradient sketches are aggregated **(3)**, and then **(4)** momentum and **(5)** error accumulation are applied to the sketch.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

that `FedAvg` cannot meaningfully reduce the overall communication cost without degrading the final accuracy.

Client dataset sizes tend to follow a power law distribution (Guo et al., 2009), so most clients are likely to have datasets that are too small to benefit from many local steps. Prior work has shown that it is necessary to train on the small datasets at the tail of the data distribution in order to create a global model that works for all clients (Smith et al., 2017). TODO: verify the reference

Furthermore, while reducing the total number of communication steps is important, each client in `FedAvg` still has to download and upload vectors the same size as the model. 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). Note that because the entire parameter vector is updated every iteration, clients must download the latest model right before local training.

Due to these limitations in a number of important settings, it may sometimes be desirable to augment or replace `FedAvg` with a technique that reduces the amount of communication required each round instead of only reducing the total number of communication rounds. A number of such gradient compression techniques have been proposed for federated learning. Gradient sparsification techniques such as local top-k transmit only the high-magnitude gradient elements (Wangni et al., 2018). These techniques can work well in practice, but they have no convergence guarantees (Lin et al., 2017). Ivkin et al. (2019a) 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).

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, specifically they apply a combination of subsampling, quantization and random rotation.

2.2 Statelessness

In order to practically train models on federated datasets of sizes in the millions, Kairouz et al. (2019) emphasize multiple times in their comprehensive survey of federated learning techniques that clients only participate a single time in federated learning, and so clients cannot have local state.

Quantization methods transmit gradients with reduced numerical precision (Agarwal et al., 2018; Alistarh et al., 2017), achieving up to 32x compression (Bernstein et al., 2018). However, unbiased quantization methods have limited empirical success. Biased gradient compression methods, such as top-k sparsification or signSGD, tend to achieve the highest accuracy. 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 (Lin et al., 2017; Karimireddy et al., 2019b). However, because clients are not expected to participate more than once in the lifetime of a federated training task, the error never has a chance to be re-introduced.

2.3 Non-I.I.D. Data

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. 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 practice, to achieve good accuracy and reduce the number of communication rounds FedAvg requires many local steps (McMahan et al., 2016). However, carrying out too many local steps risks model drift due to the non-IID nature of local datasets (Wang et al., 2019a; Karimireddy et al., 2019a). Variants of FedAvg have been proposed which employ proximal updates (Sahu et al., 2018) or variance reduction techniques (Karimireddy et al., 2019a) to mitigate model drift. However, these methods require additional communication, and are empirically untested.

In this work we propose FedSketchedSGD, a gradient compression algorithm that conforms to all three of these constraints at once. FedSketchedSGD achieves communication efficiency on non-i.i.d data without requiring any local state, and we prove that it converges under additional assumptions (See Theorem 2). FedSketchedSGD can also carry out a computation that is equivalent to performing momentum on the local gradients before compression, again without any maintaining local state.

3 FedSketchedSGD

3.1 Federated Learning Setup

In federated learning there are C clients with samples D_i drawn i.i.d. from distinct unknown data distributions $\{\mathcal{D}_i\}$. We do not assume that $\mathcal{D}_i$ 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:

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

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

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

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).

3.2 Algorithm

In the federated setting, a central aggregator coordinates an iterative optimization procedure to minimize f with respect to the parameters of a model. In every iteration, the aggregator chooses W clients uniformly at random, and these clients download the current model.[3]

The i^{th} client computes a stochastic gradient g_t^i using a batch of its local data, then compresses g_t^i using a data structure called a Count Sketch. 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 recovered. We provide more details on the Count Sketch in Appendix C, but here we treat it simply as a compression operator, with the special property that it is linear. After computing g_t^i, each worker computes the sketch $S(\mathrm{g}_t^i)$ and uploads it over the Internet to the aggregator.

In other biased gradient compression schemes, including (Ivkin et al., 2019a), which also uses sketches for gradient compression, the participating workers compute and store locally the error introduced by the compression method; the error is added back to the gradient the next time the client participates. This local error feedback is important, both to converge theoretically and to achieve high performance in practice. Momentum (u_t^i) also takes place on the workers, because otherwise the momentum would only operate on the portions of the gradient that survived compression. For some compression operator $\mathcal{C}$ with decompression operator $\mathcal{D}$, and given a learning rate η and momentum ρ, the computation of a typical biased gradient compression scheme is as follows:

$$\begin{aligned} \mathrm{u}_{t+1}^i &= \rho \mathrm{u}_t^i + \mathrm{g}_t^i \\ \mathrm{e}_{t+1}^i &= \mathrm{e}_t^i + \eta \mathrm{u}_t^i - \mathcal{D}(\mathcal{C}(\mathrm{u}_t^i + \mathrm{e}_t^i))) \\ \mathrm{w}_{t+1} &= \mathrm{w}_t - \mathcal{D}\left(\frac{1}{W} \sum_{i=1}^{W} \mathcal{C}(\mathrm{u}_t^i + \mathrm{e}_t^i) \right) \end{aligned} \tag{3}$$

Note that the momentum and error accumulation vectors u_t^i and e_t^i are both local state. However, the federated setting does not allow for local state, since clients generally only participate once in all of training (Kairouz et al., 2019). FedSketchedSGD circumvents this limitation by taking advantage of the linearity of the Count Sketch: the server maintains a single error accumulation sketch, which, because the Count Sketch is linear, is equivalent to carrying out error feedback separately on each client and summing the results. In more detail, given sketching and unsketching operators $\mathcal{S}$ and $\mathcal{U}$, the aggregator collects and averages the sketches, and then carries out momentum and error feedback. Finally, the aggregator updates the model using k heavy coordinates extracted from the accumulated sketches:

$$\begin{aligned} S(\mathrm{g}_t) &= \frac{1}{W} \sum_{i=1}^{W} \mathcal{S}(\mathrm{g}_t^i) \\ S(\mathrm{u}_{t+1}) &= \rho S(\mathrm{u}_t) + S(\mathrm{g}_t) \\ \Delta &= \text{Top-k}(\mathcal{U}(\eta S(\mathrm{u}_{t+1}) + S(\mathrm{e}_t))) \\ S(\mathrm{e}_{t+1}) &= \eta S(\mathrm{u}_{t+1}) + S(\mathrm{e}_t) - \mathcal{S}(\Delta) \\ \mathrm{w}_{t+1} &= \mathrm{w}_t - \Delta \end{aligned}$$

FedSketchedSGD is presented in full in Algorithm 1.

4 Theory

This section presents convergence guarantees for FedSketchedSGD. First, Section 4.1 gives the convergence of FedSketchedSGD when making a strong and

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

Algorithm 1 `FedSketchedSGD`

Input: k, T, η, momentum parameter ρ, local batch size ℓ
Input: Client datasets $\{D_i\}_{i=1}^{C}$
Input: Number of clients selected per round W
Input: Initial model weights w_0
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
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_t^i on batch B_i of size ℓ: $g_t^i = \frac{1}{\ell}\sum_{i=1}^{l} \nabla_w \mathcal{L}(w^t, x_i)$
    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 $w_{t+1} = w_t - \Delta$
end for
Output: $\{w_t\}_{t=1}^{T}$

opaque assumption on the sequence of gradients. Section 4.2 instead makes a more interpretable assumption on the gradients, and arrives at a weaker convergence guarantee.

4.1 Alternative 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 heavy hitters 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, `FedSketchedSGD` never stores 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. *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. *For an L-smooth non-convex function f, `FedSketchedSGD`, with step size $\eta = \frac{1-\rho}{2L\sqrt{T}}$, with stochastic gradients of norm bounded by G, under Assumption 1, 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 size of the sketch communicated from each participating client to the parameter server is $\mathcal{O}(k \log(T/\delta))$ per round.*

Note that the contraction factor τ should be thought as a function of k, and therefore highlights the tradeoff 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 isn't immediately obvious that we should expect it to hold. To remedy this, the next section analyzes `FedSketchedSGD` under a simpler assumption that involves only the gradients. Note that this is still an assumption on the algorithmic path, but this presents a clearer understanding.

4.2 Alternative 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:

Enayat: correct definition: remove almost surely TODO: verify if we can get rid of one of alphas

Definition 1. *[(I, α_1, α_2)-sliding heavy]*
A stochastic process $\{g_t\}_t$ is (I, α)-sliding heavy if at any moment t gradient vector g_t can be decomposed as: $g_t = g_t^N + g_t^S$, where g_t^S is a signal and g_t^N is noise with the following properties:

1. ***[Signal]** For every non-zero coordinate j of vector g_t^S: $\exists t_1 \leq t \leq t_2,\ t_1 - t_2 \leq I :\ |\sum_{t_1}^{t_2} g_t^j| > \alpha_1 \|\sum_{t_1}^{t_2} g_t\|$ a.s.*
2. ***[Noise]** $\forall i \in \{i\}_{t-I'}^{t}, g_t^N$ is mean zero, symmetric, potentially dense and when normalized by its norm, its second moment bounded as $\|g_t^N\| \leq \alpha_2 \|g_t\|$ verify that martingale property does not break*

The above definition requires gradients to have heavy coordinates relative to its norm, so as gradients become smaller

as optimization proceeds, the signal and noise adjusts to the norm. This definition reduces to the stronger assumption that gradients always contain heavy coordinates when $I = 1$. 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. Note that the existence of heavy coordinates that are spread along a sequence of I updates also explains the success of error feedback techniques, which extract signal from a sequence of gradients that is indistinguishable from noise in any one iteration.

To see why this assumption helps to show that `FedSketchedSGD` converges, consider a sequence of gradients shown in Figure 2. Under an (I, α)-sliding heavy assumption on the gradients, the signal that captures most of the mass of the gradients takes the form of heavy hitters in sums of consecutive g_t. To capture this signal, we could maintain I error accumulation sketches $\{\mathrm{e}_j\}_{j=1}^{I}$, where each sketch accumulates gradients starting at a different g_i. Then, at any given iteration, we could consult all the error accumulation sketches to find if there were any signal spread across the previous $I' < I$ iterations. Once this signal is identified, the assumption guarantees that the only other component of the sequence of gradients is noise, which can be accounted for in the convergence proof.

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). To address the task of identifying heavy hitters that are spread over a sequence of gradients, it is possible to use a sliding window Count Sketch, which is a data structure that can locate these heavy hitters using only $\log(I)$ error accumulation sketches. Additional details on sliding window Count Sketch are in Appendix D.

We now describe the sliding window error accumulation data structure S_t. The data structure is essentially a function of the last I gradients, and has a function $\texttt{FindHeavy}_\alpha$, which returns a vector C_t with α- 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.

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

Using this, we show `FedSketchedSGD` converges:

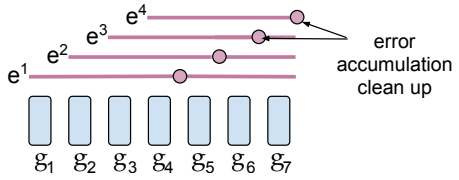


Figure 2: Sliding window error accumulation

Theorem 2. *For an L-smooth [4] non-convex function $f = \widehat{\mathbb{E}} f_i$, `FedSketchedSGD`, with step size $\eta = \frac{1}{T^{2/3}}$ and $\rho = 0$ (no momentum), using stochastic gradients g_i of f_i such that $\mathbb{E}\|\mathrm{g}_i\|^2 \leq G_i^2$, and let $G^2 := \frac{\sum_{i=1}^{C} G_i^2}{C}$, then under the assumption that gradients form a (I, α)-sliding heavy stochastic process, in T iterations, with probability at least $1 - \delta$, returns $\{w_t\}_{t=1}^{T}$ such that*

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

Remarks:

1. These guarantee are for the non-i.i.d. setting – i.e. f is the average test error with respect to potentially unrelated distributions (see eqn. 2).
2. We see that the convergence in Theorem 1 matches that of uncompressed SGD, however 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).

5 Evaluation

We implement and compare `FedSketchedSGD`, gradient sparsification (local top-k), and `FedAvg`. All methods are implemented in PyTorch (Paszke et al., 2019). Our implementation of `FedSketchedSGD` uses a vanilla Count Sketch as a compression operator, and for all methods, we employ momentum factor masking, following (Lin et al., 2017).

One key advantage of `FedSketchedSGD` over gradient sparsification methods is that the compression operator is linear, meaning that executing a step using only a single client with N data points is equivalent to executing a step using N clients, each of which have only a single data point. Gradient sparsification methods do not have this property, and so we expect them to degrade as the size of local datasets decreases and the number participating clients increases.

Smaller local datasets and more non-i.i.d. data also hinder `FedAvg`, since taking many local steps on only a few

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

data points leads quickly to local overfitting. We therefore see the regime of small, non-i.i.d. local datasets and many participating workers as a relatively unsolved regime in federated learning. 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.

For each method, we measure the number of bytes each client needs to upload or download,[5] and we report compression factors with respect to the total upload/download required when using SGD. One important consideration not captured in these numbers is that in `FedAvg`, clients must download all the new model parameters right before participating, because every model weight could get updated in every round. In contrast, local top-k and `FedSketchedSGD` 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. Because of this feature, upload compression is much more important for local top-k and `FedSketchedSGD` than download compression. Download compression is also less important for all three methods, because residential Internet connections tend to be asynchronous, with far higher download speed than upload (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 figures in the appendix.

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 the appendix. `FedAvg` achieves compression by reducing the number of iterations carried out, so for these runs, we simply scale the learning rate schedule 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 `FedSketchedSGD` we adjust k and the number of columns in 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 do not find that `FedAvg` converges to the baseline accuracy for even a small number of local epochs, where the learning rate decay has the smallest effect. Since any decay would have no time to take effect, we do not employ any decay.

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 non-i.i.d. datasets by giving each client images from only a single class. For CIFAR10 we use 10,000 clients, yielding 5 images per client, and for CIFAR100, 50,000 clients, yielding 1 image per client. Our 7M-parameter model architecture, data preprocessing, and most hyperparameters follow (Page, 2019). Details can be found in the Appendix. We report the final accuracy of the trained model 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 with global momentum both fail to converge. For CIFAR10, `FedAvg` does not converge when using more than a single local iteration (using all 5 data points in each iteration), and for CIFAR100, only one `FedAvg` run converges (with 2 local iterations). 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.

Although local top-k is able to achieve comparable accuracy to the uncompressed baseline for a number of parameter settings, the downloaded update is essentially dense since the workers are updating different coordinates. This reduces the download compression to $1\times$, capping the overall compression at $2\times$.

5.2 FEMNIST (ResNet101)

The experiments above show that `FedSketchedSGD` significantly outperforms competing methods in the regime we expect it to excel at. In this section we introduce a task designed to be more favorable for `FedAvg`, and show that `FedSketchedSGD` still performs competitively.

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 in the Appendix. We report the final accuracy of the trained model 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 (200 images per client). The clients are split according to the person who wrote the character, so the data is less non-i.i.d. than our per-class splits of CIFAR10. Despite this, `FedSketchedSGD` performs competitively with both `FedAvg` and local top-k, as shown in Figure 5.

In this learning task, no local state is possible, since clients participate only a single time. Therefore, as expected, `FedAvg`, which relies on no local state, outperforms local

[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.

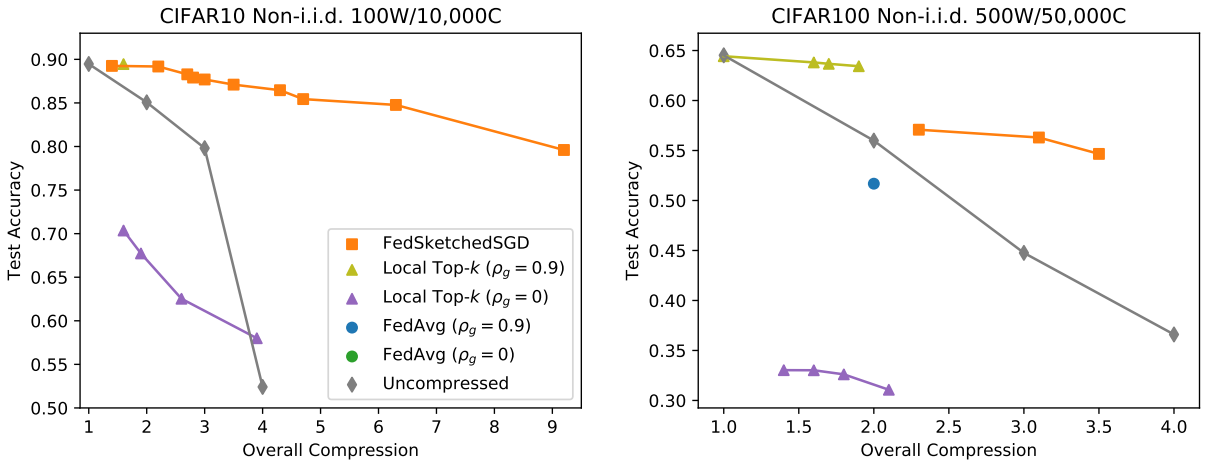


Figure 3: Left: CIFAR10 with 10,000 clients each containing 5 images from one class. "Uncompressed" refers to runs that attain compression by simply running for fewer epochs (equivalent to federated averaging with 1 local iteration on the entire dataset). `FedSketchedSGD` outperforms all methods. Right: CIFAR100 with 50,000 clients each containing 1 image. Local top-k cannot achieve high compression, but sketching can. Only one `FedAvg` run using more than a single local step converges.

top-k (with 0 global momentum), which typically requires error feedback. 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 the other tasks we experiment with, 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 each 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 4.

Figure 4 plots loss curves (negative log likelihood) achieved during training. 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. We show final perplexity numbers for these runs in Figure 4.

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. Loss curves for `FedAvg` or global momentum local top-k, which did not perform well in any of the range of hyperparameters we tested, can be found in the Appendix. 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 decreasing the amount that needs to be communicated each round. In this work, we complement this body of work by introducing `FedSketchedSGD`, 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 `FedSketchedSGD` 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.

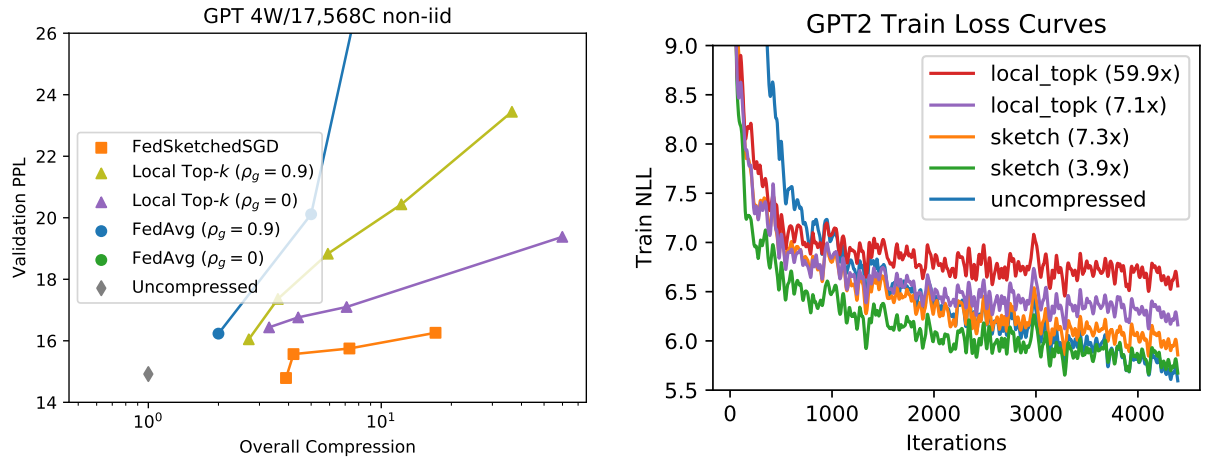


Figure 4: Left: `FedSketchedSGD` achieves lossless compression compared to uncompressed, and consistently achieves lower perplexity than top-k runs with the same compression. Right: Training loss curves

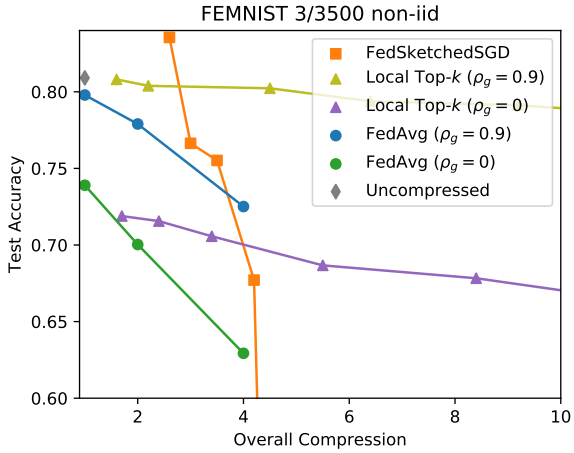


Figure 5: `FedSketchedSGD` is competitive with FedAvg and local top-k for lower compression.

References

Agarwal, N., Suresh, A. T., Yu, F., Kumar, S., and Mcmahan, H. B. (2018). cpsgd: Communication-efficient and differentially-private distributed sgd.

Alistarh, D., Grubic, D., Li, J., Tomioka, R., and Vojnovic, M. (2017). Qsgd: Communication-efficient sgd via gradient quantization and encoding. In *Advances in Neural Information Processing Systems*, pages 1709–1720.

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

Bagdasaryan, E., Veit, A., Hua, Y., Estrin, D., and Shmatikov, V. (2018). How to backdoor federated learning.

Bernstein, J., Wang, Y.-X., Azizzadenesheli, K., and Anandkumar, A. (2018). signsgd: Compressed optimisation for non-convex problems. *arXiv preprint arXiv:1802.04434*.

Bhagoji, A. N., Chakraborty, S., Mittal, P., and Calo, S. (2018). Analyzing federated learning through an adversarial lens. *arXiv preprint arXiv:1811.12470*.

Bonawitz, K., Ivanov, V., Kreuter, B., Marcedone, A., McMahan, H. B., Patel, S., Ramage, D., Segal, A., and Seth, K. (2016). Practical secure aggregation for federated learning on user-held data. *arXiv preprint arXiv:1611.04482*.

Braverman, V., Chestnut, S. R., Ivkin, N., Nelson, J., Wang, Z., and Woodruff, D. P. (2017). 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.

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

Brisimi, T. S., Chen, R., Mela, T., Olshevsky, A., Paschalidis, I. C., and Shi, W. (2018). Federated learning of predictive models from federated electronic health records. *International journal of medical informatics*, 112:59–67.

Caldas, S., Duddu, S. M. K., Wu, P., Li, T., Konecny, J., McMahan, H. B., Smith, V., and Talwalkar, A. (2018). Leaf: A benchmark for federated settings.

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

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

Datar, M., Gionis, A., Indyk, P., and Motwani, R. (2002). Maintaining stream statistics over sliding windows. *SIAM journal on computing*, 31(6):1794–1813.

Dean, J., Corrado, G., Monga, R., Chen, K., Devin, M., Mao, M., Ranzato, M., Senior, A., Tucker, P., Yang, K., et al. (2012). Large scale distributed deep networks. In *Advances in neural information processing systems*, pages 1223–1231.

EU (2018). 2018 reform of eu data protection rules.

Geyer, R. C., Klein, T., and Nabi, M. (2017). Differentially private federated learning: A client level perspective.

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

Goyal, P., Dollár, P., Girshick, R., Noordhuis, P., Wesolowski, L., Kyrola, A., Tulloch, A., Jia, Y., and He, K. (2017). Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*.

Guo, L., Tan, E., Chen, S., Zhang, X., and Zhao, Y. (2009). Analyzing patterns of user content generation in online social networks. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 369–378.

Hard, A., Rao, K., Mathews, R., Ramaswamy, S., Beaufays, F., Augenstein, S., Eichner, H., Kiddon, C., and Ramage, D. (2018). Federated learning for mobile keyboard prediction. *arXiv preprint arXiv:1811.03604*.

Hardy, S., Henecka, W., Ivey-Law, H., Nock, R., Patrini, G., Smith, G., and Thorne, B. (2017). Private federated learning on vertically partitioned data via entity resolution and additively homomorphic encryption. *arXiv preprint arXiv:1711.10677*.

Ivkin, N., Liu, Z., Yang, L. F., Kumar, S. S., Lemson, G., Neyrinck, M., Szalay, A. S., Braverman, V., and Budavari, T. (2018). Scalable streaming tools for analyzing n-body simulations: Finding halos and investigating excursion sets in one pass. *Astronomy and computing*, 23:166–179.

Ivkin, N., Rothchild, D., Ullah, E., Stoica, I., Arora, R., et al. (2019a). Communication-efficient distributed sgd with sketching. In *Advances in Neural Information Processing Systems*, pages 13144–13154.

Ivkin, N., Yu, Z., Braverman, V., and Jin, X. (2019b). Qpipe: Quantiles sketch fully in the data plane. In *Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies*, pages 285–291.

Kairouz, P., McMahan, H. B., Avent, B., Bellet, A., Bennis, M., Bhagoji, A. N., Bonawitz, K., Charles, Z., Cormode, G., Cummings, R., D'Oliveira, R. G. L., Rouayheb, S. E., Evans, D., Gardner, J., Garrett, Z., Gascón, A., Ghazi, B., Gibbons, P. B., Gruteser, M., Harchaoui, Z., He, C., He, L., Huo, Z., Hutchinson, B., Hsu, J., Jaggi, M., Javidi, T., Joshi, G., Khodak, M., Konecny, J., Korolova, A., Koushanfar, F., Koyejo, S., Lepoint, T., Liu, Y., Mittal, P., Mohri, M., Nock, R., Özgür, A., Pagh, R., Raykova, M., Qi, H., Ramage, D., Raskar, R., Song, D., Song, W., Stich, S. U., Sun, Z., Suresh, A. T., Tramèr, F., Vepakomma, P., Wang, J., Xiong, L., Xu, Z., Yang, Q., Yu, F. X., Yu, H., and Zhao, S. (2019). Advances and open problems in federated learning.

Karimireddy, S. P., Kale, S., Mohri, M., Reddi, S. J., Stich, S. U., and Suresh, A. T. (2019a). Scaffold: Stochastic controlled averaging for on-device federated learning.

Karimireddy, S. P., Rebjock, Q., Stich, S. U., and Jaggi, M. (2019b). Error feedback fixes signsgd and other gradient compression schemes. *arXiv preprint arXiv:1901.09847*.

Konecny, J., McMahan, H. B., Yu, F. X., Richtárik, P., Suresh, A. T., and Bacon, D. (2016). Federated learning: Strategies for improving communication efficiency.

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

Lee, K., Lee, J., Yi, Y., Rhee, I., and Chong, S. (2010). Mobile data offloading: How much can wifi deliver? In *Proceedings of the 6th International COnference*, pages 1–12.

Leroy, D., Coucke, A., Lavril, T., Gisselbrecht, T., and Dureau, J. (2019). Federated learning for keyword spotting. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6341–6345. IEEE.

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

Li, T., Liu, Z., Sekar, V., and Smith, V. (2019). Privacy for free: Communication-efficient learning with differential privacy using sketches. *arXiv preprint arXiv:1911.00972*.

Lin, T., Stich, S. U., Patel, K. K., and Jaggi, M. (2018). Don't use large mini-batches, use local sgd.

Lin, Y., Han, S., Mao, H., Wang, Y., and Dally, W. J. (2017). Deep gradient compression: Reducing the communication bandwidth for distributed training. *arXiv preprint arXiv:1712.01887*.

Liu, Z., Ivkin, N., Yang, L., Neyrinck, M., Lemson, G., Szalay, A., Braverman, V., Budavari, T., Burns, R., and Wang, X. (2015). Streaming algorithms for halo finders. In *2015 IEEE 11th International Conference on e-Science*, pages 342–351. IEEE.

McMahan, H. B., Moore, E., Ramage, D., Hampson, S., et al. (2016). Communication-efficient learning of deep networks from decentralized data. *arXiv preprint arXiv:1602.05629*.

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

Muchnik, L., Pei, S., Parra, L. C., Reis, S. D., Andrade Jr, J. S., Havlin, S., and Makse, H. A. (2013). Origins of power-law degree distribution in the heterogeneity of human activity in social networks. *Scientific reports*, 3(1):1–8.

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

Nishio, T. and Yonetani, R. (2019). Client selection for federated learning with heterogeneous resources in mobile edge. In *ICC 2019-2019 IEEE International Conference on Communications (ICC)*, pages 1–7. IEEE.

Page, D. (2019). How to train your resnet.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In Wallach, H., Larochelle, H., Beygelzimer, A., dAlché Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. (2019). Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8):9.

Robbins, H. and Monro, S. (1951). A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407.

Sahu, A. K., Li, T., Sanjabi, M., Zaheer, M., Talwalkar, A., and Smith, V. (2018). On the convergence of federated optimization in heterogeneous networks. *arXiv preprint arXiv:1812.06127*.

Sattler, F., Wiedemann, S., Müller, K.-R., and Samek, W. (2019). Robust and communication-efficient federated learning from non-iid data. *IEEE transactions on neural networks and learning systems*.

Shi, S., Chu, X., Cheung, K. C., and See, S. (2019). Understanding top-k sparsification in distributed deep learning. *arXiv preprint arXiv:1911.08772*.

Shi, W., Cao, J., Zhang, Q., Li, Y., and Xu, L. (2016). Edge computing: Vision and challenges. *IEEE internet of things journal*, 3(5):637–646.

Smith, V., Chiang, C.-K., Sanjabi, M., and Talwalkar, A. S. (2017). Federated multi-task learning. In *Advances in Neural Information Processing Systems*, pages 4424–4434.

Stich, S. U. (2018). Local sgd converges fast and communicates little. *arXiv preprint arXiv:1805.09767*.

Stich, S. U., Cordonnier, J.-B., and Jaggi, M. (2018). Sparsified sgd with memory. In *Advances in Neural Information Processing Systems*, pages 4447–4458.

Sutskever, I., Martens, J., Dahl, G., and Hinton, G. (2013). On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147.

Tomlinson, M., Solomon, W., Singh, Y., Doherty, T., Chopra, M., Ijumba, P., Tsai, A. C., and Jackson, D. (2009). 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.

Wang, J. and Joshi, G. (2018). Cooperative sgd: A unified framework for the design and analysis of communication-efficient sgd algorithms.

Wang, J., Sahu, A. K., Yang, Z., Joshi, G., and Kar, S. (2019a). Matcha: Speeding up decentralized sgd via matching decomposition sampling. *arXiv preprint arXiv:1905.09435*.

Wang, J., Tantia, V., Ballas, N., and Rabbat, M. (2019b). Slowmo: Improving communication-efficient distributed sgd with slow momentum. *arXiv preprint arXiv:1910.00643*.

Wang, S., Tuor, T., Salonidis, T., Leung, K. K., Makaya, C., He, T., and Chan, K. (2019c). Adaptive federated learning in resource constrained edge computing systems. *IEEE Journal on Selected Areas in Communications*, 37(6):1205–1221.

Wangni, J., Wang, J., Liu, J., and Zhang, T. (2018). Gradient sparsification for communication-efficient distributed optimization. In *Advances in Neural Information Processing Systems*, pages 1299–1309.

Wolf, T. (2019). How to build a state-of-the-art conversational ai with transfer learning.

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

Yang, L. T., Augustinus, B., Ma, J., Tan, L., and Srinivasan, B. (2010). *Mobile intelligence*, volume 69. Wiley Online Library.

Yang, T., Andrew, G., Eichner, H., Sun, H., Li, W., Kong, N., Ramage, D., and Beaufays, F. (2018). Applied federated learning: Improving google keyboard query suggestions. *arXiv preprint arXiv:1812.02903*.

Zhang, S., Dinan, E., Urbanek, J., Szlam, A., Kiela, D., and Weston, J. (2018). Personalizing dialogue agents: I have a dog, do you have pets too?

Zhao, Y., Li, M., Lai, L., Suda, N., Civin, D., and Chandra, V. (2018). Federated learning with non-iid data.

Zheng, S., Huang, Z., and Kwok, J. (2019). Communication-efficient distributed blockwise momentum sgd with error-feedback. In *Advances in Neural Information Processing Systems*, pages 11446–11456.

The appendix is organized as follows:

- Appendix A lists hyperparameters and model architectures used in all experiments, and includes more plots with additional experimental data, including results broken down into upload, download and overall compression.
- Appendix B gives full proofs of convergence for `FedSketchedSGD`.
- Appendix C describes the Count Sketch data structure and how it is used in `FedSketchedSGD`.
- 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. When the uncompressed baseline does not converge, we decrease the maximum learning rate of the triangular learning rate schedule from 0.4 to 0.3 for CIFAR10, and from 0.4 to 0.2 for CIFAR100, for which the uncompressed baseline converges. We then use this learning rate schedule for all compressed runs. When `FedAvg` runs fewer than 24 epochs, we compress the learning rate schedule accordingly. Although we could further tune the learning rate schedule for `FedAvg` in order to achieve convergence, we restrict ourselves to not tuning the learning rate schedule for any of the other methods, as this would add yet another hyperparameter to be searched over. We use a global batch size of 500, which corresponds to 1-5 local datapoints per client depending on the setting. During training, we do further preprocessing and augmentation of the data via image transformations, specifically a random crop and random horizontal flip. At both train and test time, we normalize the images according to the empirical mean and standard deviation. 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. Most of these training procedures, and the modified ResNet9 architecture we use, are drawn from the work of Page (2019).

`FedSketchedSGD`, `FedAvg` and local top-k each have unique hyperparameters which we need to search over. For `FedSketchedSGD` we try a grid of values for k and the number of columns. For k we try values of [10000, 25000, 50000, 75000, 100000]. For the number of columns we try values of [325000, 650000, 1300000, 2000000, 3000000]. We also tune k for local top-k, trying values of [325000, 650000, 1300000, 2000000, 3000000, 5000000]. 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, 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 number of local epochs to perform, and the local batch size. We run a gridsearch over local epochs of [1,2,5,10] and local batch sizes of [1,5]. In order to compress `FedAvg` we need to run for fewer epochs while tuning the amount of local computation that is done. We run for [6, 8, 12, 24] epochs which correspond to $4\times, 3\times, 2\times, 1\times$ compression respectively.

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 `FedSketchedSGD` run, 3 local top-k runs, and 6 `FedAvg` runs diverged. For CIFAR100, 1 local top-k run and 5 `FedAvg` runs 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 preprocess 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 samples for training and 80,182 samples for validation over 3,500 clients. [6]

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

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

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.

`FedSketchedSGD`, `FedAvg` and local top-k each have unique hyperparameters which we need to search over. For `FedSketchedSGD` we gridsearch values for k and the number of columns. For k we use values of [used, 100000, 200000]. For the number of columns we try values of [1000000, 2000000, 5000000, 10000000]. We also tune k for local top-k, trying values of [10000, 20000, 50000, 100000, 200000, 500000, 1000000, 2000000, 5000000, 10000000, 20000000]. We present results for local top-k with and without global momentum, but not with local momentum, because each client only participates once. The two hyperparameters of interest in `FedAvg` are the number of local epochs to perform, and the local batch size. We gridsearch over local epochs of [1,2,5,10] and local batch sizes of [5,10,20,50]. In order to compress `FedAvg` we need to run for fewer epochs while tuning the amount of local computation that is done. We run for [0.25, 0.5, 1.0] epochs which correspond to $4\times, 2\times, 1\times$ compression respectively. 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.

FEMNIST is naturally non-i.i.d., but it has relatively few clients, each of which have relatively large local datasets. We are interested in this more challenging case, where each client has a very small local dataset, and many more clients participate. To simulate this, we divide each FEMNIST client into up to 40 smaller clients, each of which contain a random subset of the original client's data. In all non-`FedAvg` experiments we train for 1 epoch, meaning that each client participates only once in all of training.

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.

`FedSketchedSGD`, `FedAvg` and local top-k each have unique hyperparameters which we need to search over. For `FedSketchedSGD` we try 6 points in a grid of values for k and the number of columns. For k we use values of [10000, 25000, 50000, 100000, 200000]. For the number of columns we use values of [1240000, 12400000]. The only hyperparameter to tune for local top-k is k, for which we use values of [50000, 200000, 1240000, 5000000]. The two hyperparameters of interest in `FedAvg` are the number of local epochs to perform, and the local batch size. We use local epochs of [2,5,10] and local batch sizes of [-1] since there is only 1 datapoint per client. In order to compress `FedAvg` we need to run for fewer epochs while tuning the amount of local computation that is done. We run for [0.1, 0.2, 0.5, 1.0] epochs which correspond to $10\times, 5\times, 2\times, 1\times$ compression respectively.

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. Local top-k consistently performs worse when using global momentum (see Figure 4), so we only include results without momentum. Local momentum is not possible, since each client participates only once.

Figure 10 shows perplexities achieved for a range of k for the true top-k method.

Figure 11 shows learning curves for all runs reported in Table 1.

Plots of perplexity vs. compression, broken down into upload, download, and overall compression, can be found in Figures 8 and 9.

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: Sketching vs Local Top-k and Fedavg on GPT2

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 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 $\mathcal{D}_i$ in 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 as,

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

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,

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

We now need to bound $\left\| \mathrm{e}_t + \frac{\eta\rho}{1-\rho} \mathrm{u}_{t-1} \right\|^2$. However, we don't ever compute or store e_t and 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 $(1-\epsilon) \left\| \mathrm{x} \right\| \leq \left\| S(\mathrm{x}) \right\| \leq (1+\epsilon) \left\| \mathrm{x} \right\|$, for a user-specified constant epsilon. Note that $\left\| S(\mathrm{e}_t + \frac{\eta\rho}{1-\rho} \mathrm{u}_{t-1}) \right\|^2 \leq 2 \left(\left\| S(\mathrm{e}_t) \right\|^2 + \left(\frac{\eta\rho}{1-\rho} \right)^2 \left\| S(\mathrm{u}_{t-1}) \right\| \right)$. We bound $\left\| S(\mathrm{u}_{t-1}) \right\|$ first:

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

For the other term, we expand it and bound as

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

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 $\leq \frac{4(1+\epsilon)^2(1-\tau)\eta^2 G^2}{\tau^2(1-\rho)^2}$.

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

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

Averaging and taking expectations gives us

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

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 `FedSketchedSGD` with no momentum

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

$$\min_{t=1\cdots T} \mathbb{E}\|f(w_t)\|^2 \leq \frac{4L(f(w_0) - f^*) + \sigma^2}{\sqrt{T}} + \frac{(1+\epsilon)^2(1-\tau)G^2}{2(1-\epsilon)^2\delta^2 T}$$

B.2 Proof of Theorem 2

B.2.1 Warm-up: No error accumulation

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

$$w_{t+1} = w_t - C(\eta g_t)$$

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

$$\tilde{w}_t = w_{t-1} - C(\eta g_{t-1}) - \sum_{i=1}^{t-1} (\eta g_i - C(\eta g_i)) = w_{t-1} - \sum_{i=1}^{t-2} (\eta g_i - C(\eta g_i)) - \eta g_{t-1} = \tilde{w}_{t-1} - \eta g_{t-1}$$

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 hitters *coordinates* from x. Since by assumption there is at least one α-heavy hitter in every gradient, our compression operator recovers all of them with high probability (say $1-\delta$), with no false positives. 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, has mean zero and is symmetric. This is because the the count sketch produces unbiased estimates of coordinates and since it uses uniformly random hash function, 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, all the estimation error and noise coordinates 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 at the heavy hitter coordinates and the noise coordinates of z_i and how they contribute to the norm. Let z_i^{H} and z_i^{NH} be the heavy hitter and non-heavy coordinates, respectively. All coordinates are either heavy or not heavy, so $\|z_i\|^2 = \|z_i^{\text{H}}\|^2 + \|z_i^{\text{NH}}\|^2$. From Lemma 2 in Charikar et al. (2002), for each bucket in the count sketch, the variance in estimation is smaller than the ℓ_2 norm of tail divided by the number of buckets b. Since the tail has mass at most $(1-\alpha)G^2$, for each bucket, we have $\mathbb{E}(z_i^{\text{H}}{}_b)^2 \leq \frac{(1-\alpha)G^2}{b}$. Now if $1/\alpha^2$ is smaller than the number of buckets (which is guaranteed by our sketch size), we get $\mathbb{E}\|z_i^{\text{H}}\|^2 \leq 1/\alpha^2\frac{(1-\alpha)G^2}{b} \leq (1-\alpha)G^2$.

Plugging this in, taking $\|\nabla f(w_t)\|$ to the left hand side, averaging and taking expectation with respect to all randomness, we get,

$$\min_{t=1\cdots T}\mathbb{E}\|\nabla f(w_t)\|^2 \leq \frac{\sum_{i=1}^T\mathbb{E}\|\nabla f(w_i)\|^2}{T} \leq \frac{(f(w_0) - f^*)}{\eta T} + \eta^2 2(1-\alpha)G^2LT + L\eta G^2$$

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

$$\min_{t=1\cdots T}\mathbb{E}\|\nabla f(w_t)\|^2 \leq \frac{(f(w_0) - f^*) + 2(1-\alpha)G^2L}{T^{1/3}} + \frac{G^2L}{T^{2/3}}$$

B.2.2 Error Accumulation Data structure

Intuition. We first give some motivation for why the (α, I) sliding heavy assumption helps. Definition 1 specifies the structure of the noise included in every gradient, s.t. that growth of 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 of the entire input, which is $\eta g_t + e_t$, while coordinates are heavy only on the interval I. For example, half way to convergence $\|e_t\|$ will grow up to $O(\sqrt{T/2})$, while coordinates heavy on interval I are bounded by $O(I) \ll \sqrt{T/2}$. To avoid this problem, we suggest to regularly clean up the error accumulation: error collected on steps $1, \ldots, t-I$ can not contain the signal that is not recovered by the moment t. However, similar argument shows that maintaining e_t over the last I updates is not sufficient due to variability in how wide is an interval of heavy coordinate spread. Therefore we suggest maintaining I windows $\{e_t^i\}_{i=1}^T$ of sizes $\{1, 2, \ldots, I\}$ and clean error accumulation every I updates correspondingly (see Figure 12a). 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 count sketch can detect the signal from $g_{t,1}$. This approach finds heavy coordinates in any suffix of updates $(g_{t-I'}, \ldots, g_t)$ for $I' < I$. Recall that we maintain all error accumulations inside the sketch, thus maintaining I windows requires I sketches. Though sketch operations are very efficient and its memory size is sublinear, 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 windows techniques and how they can help to reduce 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 $S(g_t)$. 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.

$$
\begin{aligned}
S_{t+1} &= \text{Insert}(S_t, \eta S(g_t)) \\
C_t &= \text{FindHeavy}(S_{t+1}) \\
w_{t+1} &= w_t - C_t \\
S_{t+1} &= \text{Update}(S_{t+1}, C_t)
\end{aligned}
$$

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

$$
\begin{aligned}
\tilde{w}_t &= w_t - (\sum_{i=1}^{t-1} \eta g_i - C_i) \\
&= w_{t-1} - C_{t-1} - (\sum_{i=1}^{t-1} \eta g_i - C_i) \\
&= w_{t-1} - \sum_{i=1}^{t-2} \eta g_i - C_i - \eta g_{t-1} \\
&= \tilde{w}_{t-1} - \eta g_{t-1}
\end{aligned}
$$

We have $\tilde{w}_t - w_t = \sum_{i=1}^{t-1} \eta g_i - C_i$.

$$
\tilde{w}_t - w_t = \sum_{i=1}^{t-1} \eta g_i^N + \sum_{i=1}^{t-1} \eta g_i^S - \sum_{i=1}^{t-1} C_i
$$

Nikita: to fix() where g_i^N is the noise and g_i^S represents signal. Note that according to the Definition 1 $\forall i \; \exists I' \leq I$ such that there is a set of heavy coordinates in the vector $\sum_{i-I'}^{i} g_i$, we denote it as $g_i^{(1)}$ in the formula above.

As shown in the warm-up case, we will argue that

$$
\tilde{w}_t - w_t = \sum_{i=1}^{t-1} \text{estimation error}_i + \text{noise}_i = \sum_{i=1}^{t-1} z_i
$$

This follows because of the property of our data structure, which recovers all the signal in the sum of last I gradients. We now similarly argue that z_i forms a martingale difference sequence and therefore we have

$$
\mathbb{E}\|\tilde{w}_t - w_t\|^2 \leq \mathbb{E}\| \sum_{i=1}^{t} \eta g_i^{(2)} + \sum_{i=1}^{t/I} \eta g_i^{(1)} - \sum_{i=1}^{t-1} C_i \| \leq 2(1-\alpha)\eta^2 G^2
$$

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

$$
\mathbb{E} f(\tilde{w}_{t+1}) \leq f(\tilde{w}_t) - \frac{\eta}{2} \|\nabla f(w_t)\|^2 + \frac{\eta L}{2} \mathbb{E}\|\tilde{w}_t - w_t\|^2 + \frac{\eta^2 L G^2}{2}
$$

Plugging this into the bound on $\mathbb{E}\|\tilde{w}_t - w_t\|$, taking $\|\nabla f(w_t)\|$ to the left hand side, averaging and taking expectation with respect to all randomness, and choosing $\eta = \frac{1}{\sqrt{T^{2/3}}}$ we get,

$$
\min_{t=1 \cdots T} \mathbb{E}\|\nabla f(w_t)\|^2 \leq \frac{(f(w_0) - f^*) + 2(1-\alpha)G^2 L}{T^{1/3}} + \frac{G^2 L}{T^{2/3}}
$$

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 last I gradient updates (i.e. $\forall I' < I$, to recover heavy coordinates of $\sum_{i=t-I'}^{t} \eta g_i$) we can maintain I sketches in the overlapping manner depicted in Figure 12a. 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 sublinear in d (a count sketch stores $O(\log(d))$ 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.

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 sublinear 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 += $h(a_i)w_i$. 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. 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).

Big advantage of count sketch for `FedSketchedSGD` is its linearity, 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). Structure of the count sketch implies high parallelization, and enjoys great speed ups when implemented on GPU Ivkin et al. (2018).

D Sliding Windows

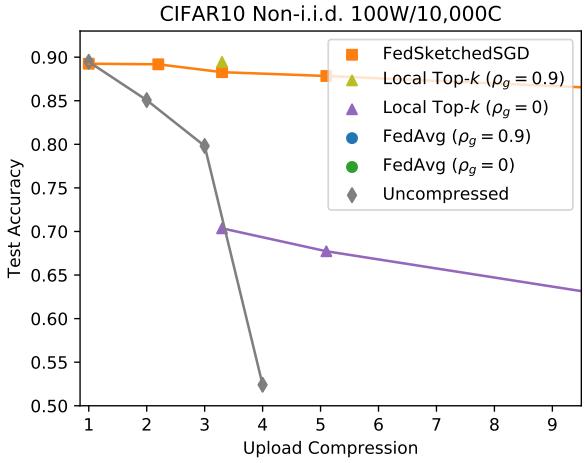
As was mentioned in Appendix C, the streaming model focuses on problems where data items arrive sequentially and their volume is too large to store on disk. In this case, accessing previous updates is prohibited, unless they are stored in

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

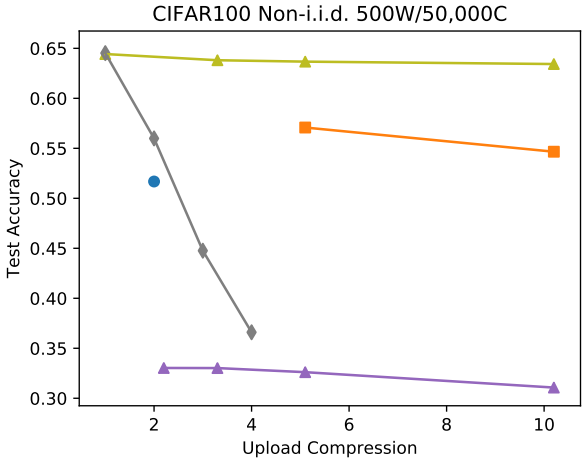
function init(r, c):
    init $r \times c$ table of counters S
    for each row r init sign and bucket hashes: $\left\{(h_j^s, h_j^b)\right\}_{j=1}^{r}$
function update((a_i, w_i)):
    for j in $1 \dots r$: $S[j, h_j^b(i)]$ += $h_j^s(i) w_i$
function estimate(i):
    init length r array estimates
 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

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 *"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. $(g_t)_{a_t} += w_t$), 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. $\widehat{\ell}_2 = \|g_t - g_{t-n}\| \pm \varepsilon\|g_t - g_{t-n}\|$.

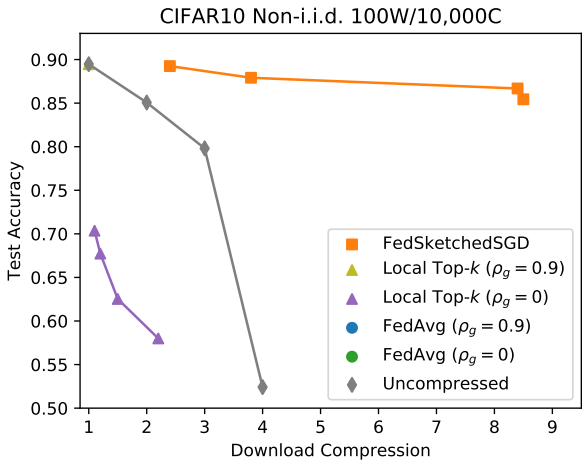
One naive solutions is to maintain n overlapping sketches, as in Fig. 12a. However, such a solution is infeasible for larger n. Currently there are 2 major frameworks to *adopt* streaming sketches to the sliding window model: exponential histograms, by Datar et al. (2002), and smooth histograms, by Braverman and Ostrovsky (2007). For simplicity, we will provide only the high level intuition behind the latter one. Maintaining all n sketches as in Fig. 12a 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 $O(\log(n))$ sketches. As in Fig. 12b, $\|S^{(i)}\| < (1+\varepsilon)\|S^{(i-1)}\|$, 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, refer to Braverman and Ostrovsky (2007).



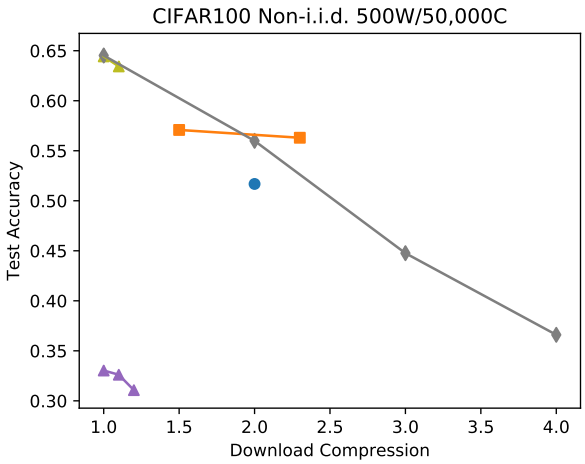
(a) CIFAR10 Upload Compression



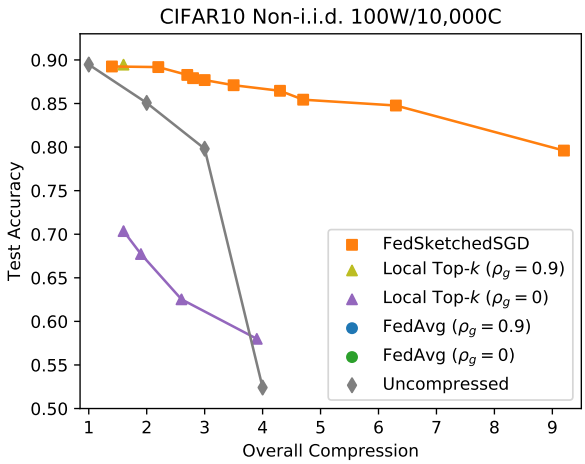
(b) CIFAR100 Upload Compression



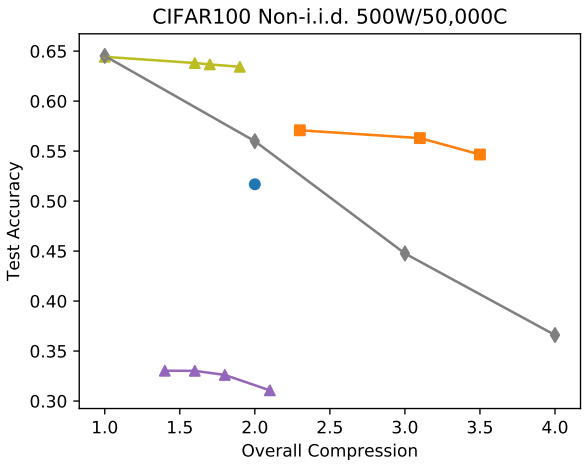
(c) CIFAR10 Download Compression



(d) CIFAR100 Download Compression

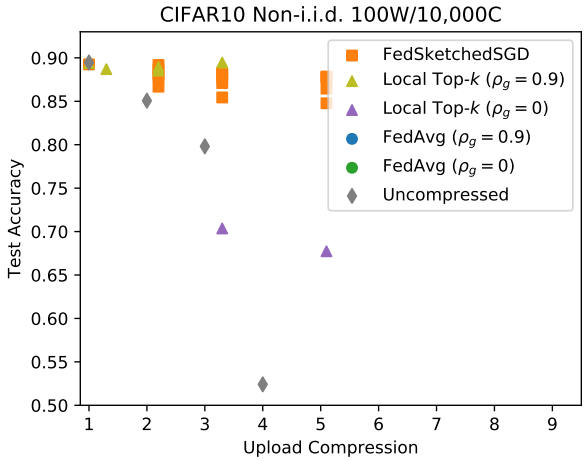


(e) CIFAR10 Overall Compression

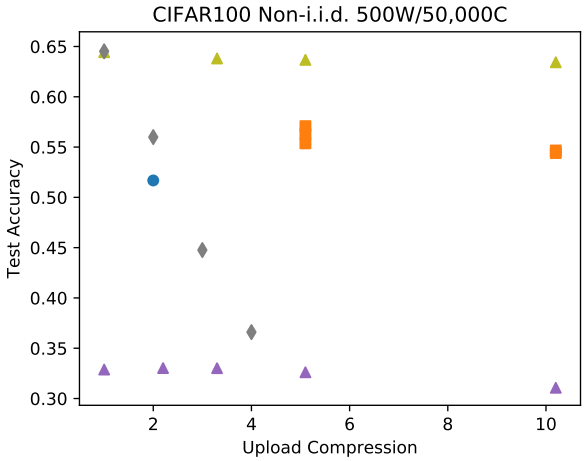


(f) CIFAR100 Overall Compression

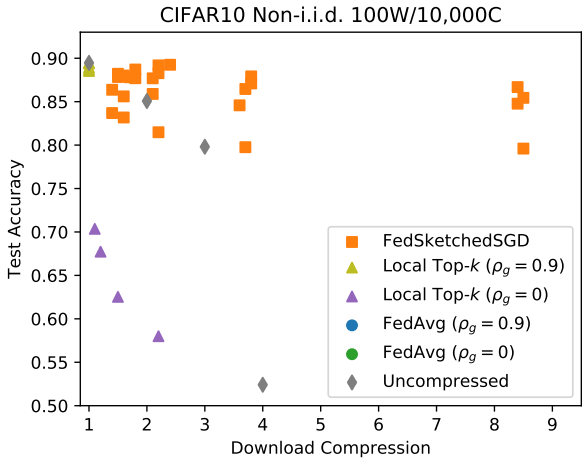
Figure 6: Upload (top), download (middle), and overall (bottom) compression for CIFAR10 (left) and CIFAR100 (right). To increase readability, each plot shows only the Pareto frontier of runs for the compression type shown in that plot. All runs that converged are shown in Figure 7.



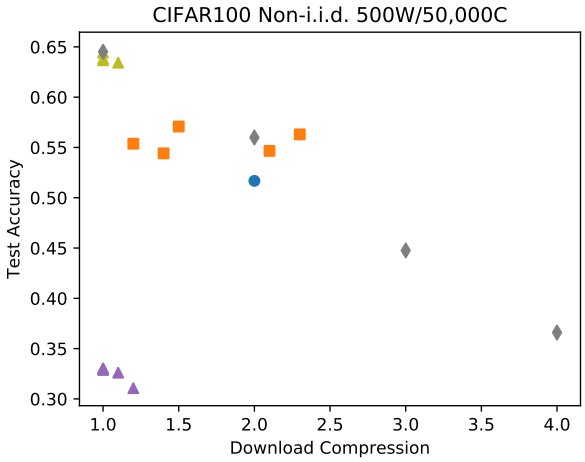
(a) CIFAR10 Upload Compression



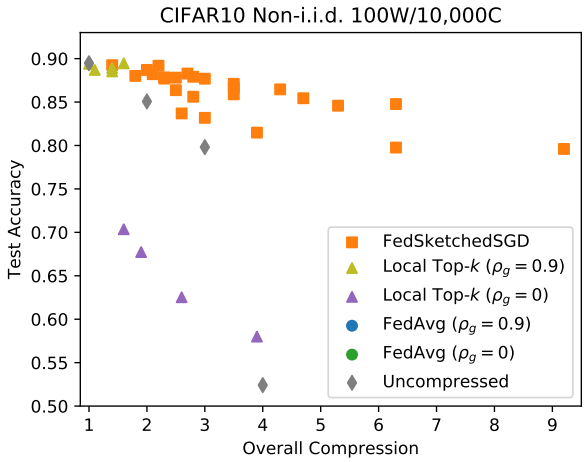
(b) CIFAR100 Upload Compression



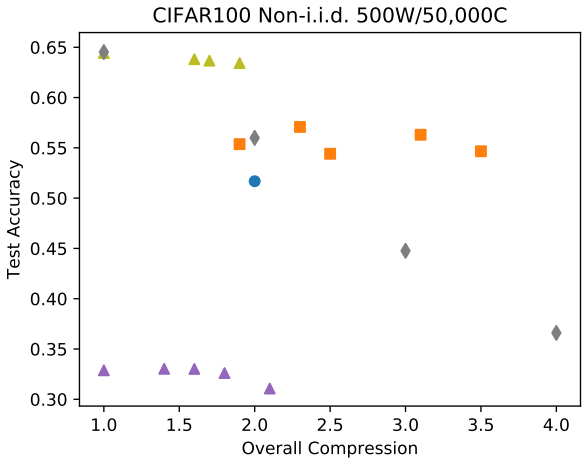
(c) CIFAR10 Download Compression



(d) CIFAR100 Download Compression

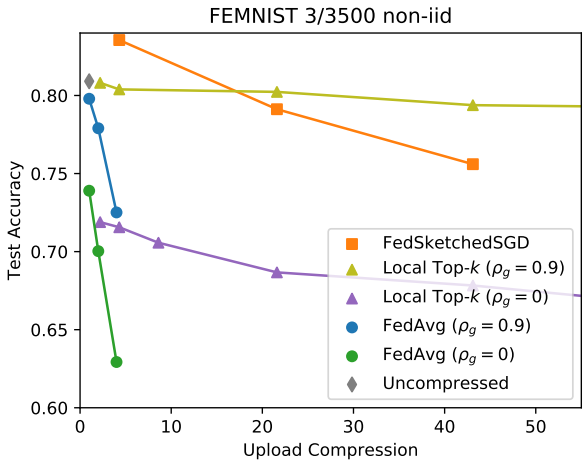


(e) CIFAR10 Overall Compression

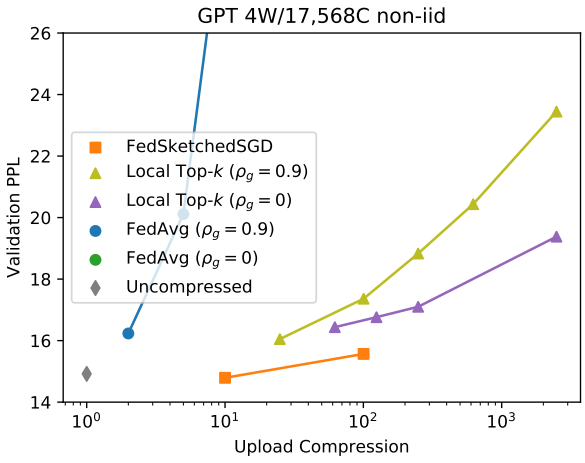


(f) CIFAR100 Overall Compression

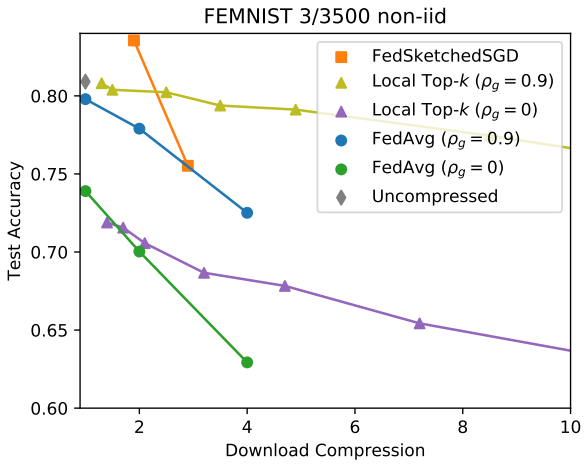
Figure 7: Upload (top), download (middle), and overall (bottom) compression for CIFAR10 (left) and CIFAR100 (right).



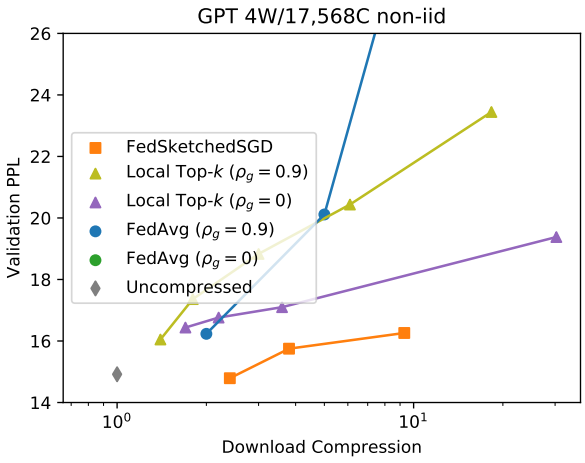
(a) FEMNIST Upload Compression



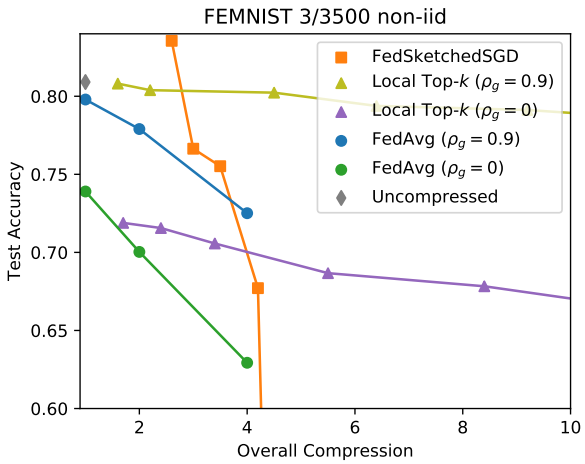
(b) PersonaChat Upload Compression



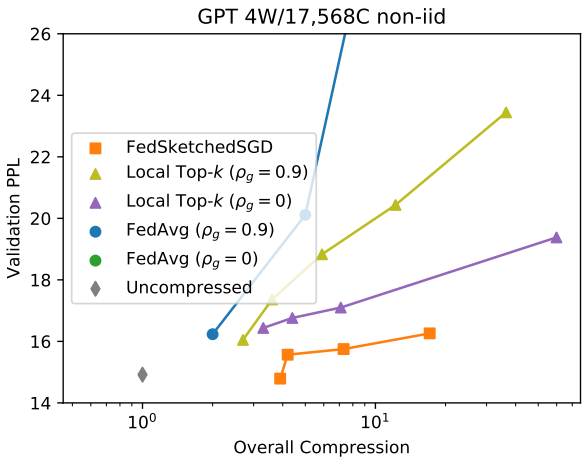
(c) FEMNIST Download Compression



(d) PersonaChat Download Compression

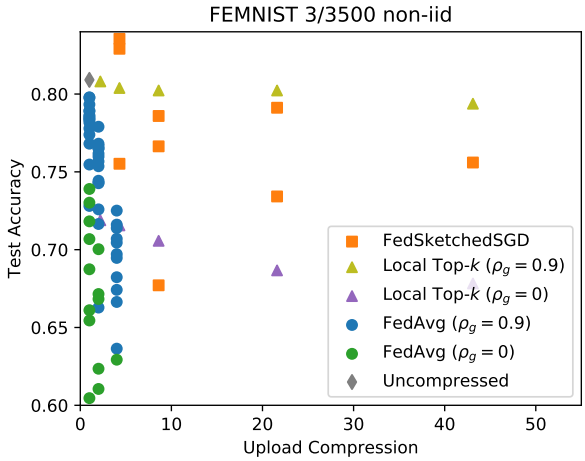


(e) FEMNIST Overall Compression

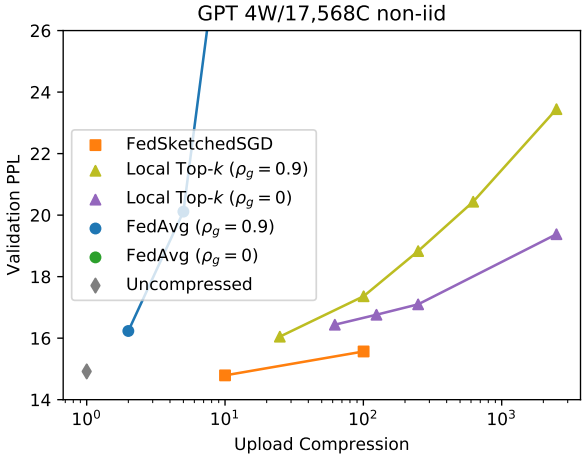


(f) PersonaChat Overall Compression

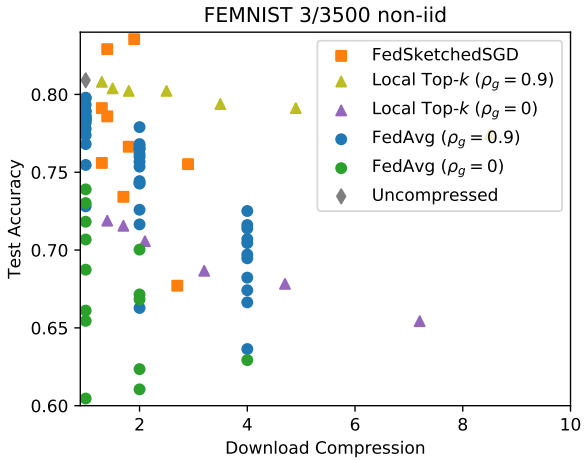
Figure 8: Upload (top), download (middle), and overall (bottom) compression for FEMNIST (left) and PersonaChat (right). To increase readability, each plot shows only the Pareto frontier of runs for the compression type shown in that plot. All results are shown in Figure 9.



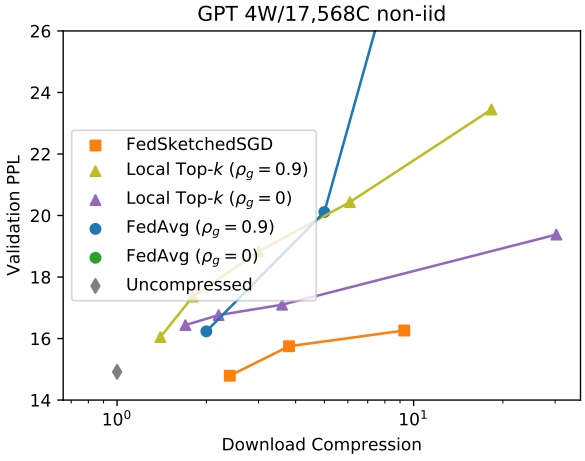
(a) FEMNIST Upload Compression



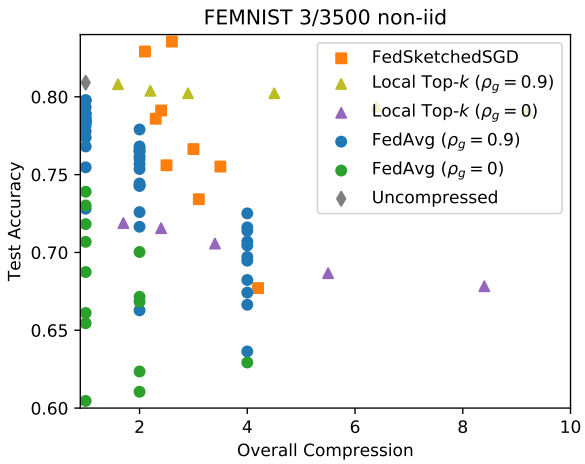
(b) PersonaChat Upload Compression



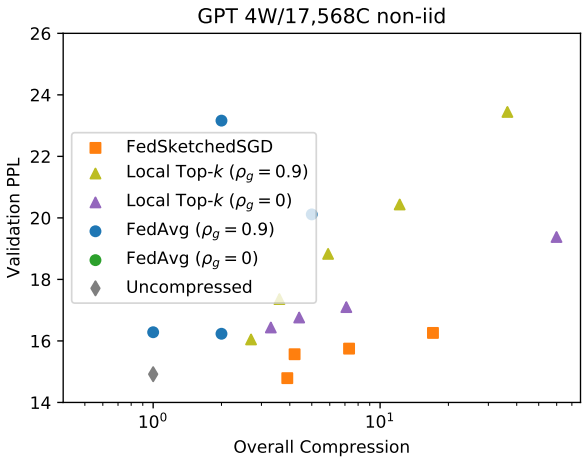
(c) FEMNIST Download Compression



(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).

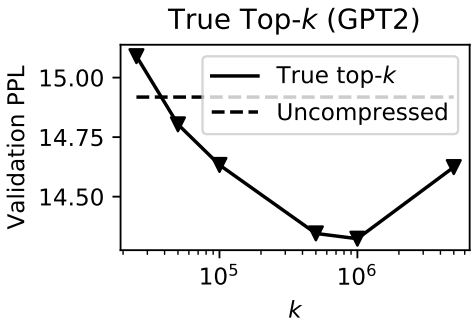


Figure 10: Validation perplexity on PersonaChat for a range of k using true top-k. For $k > 10^6$ the use of momentum factor masking in virtual momentum degrades performance.

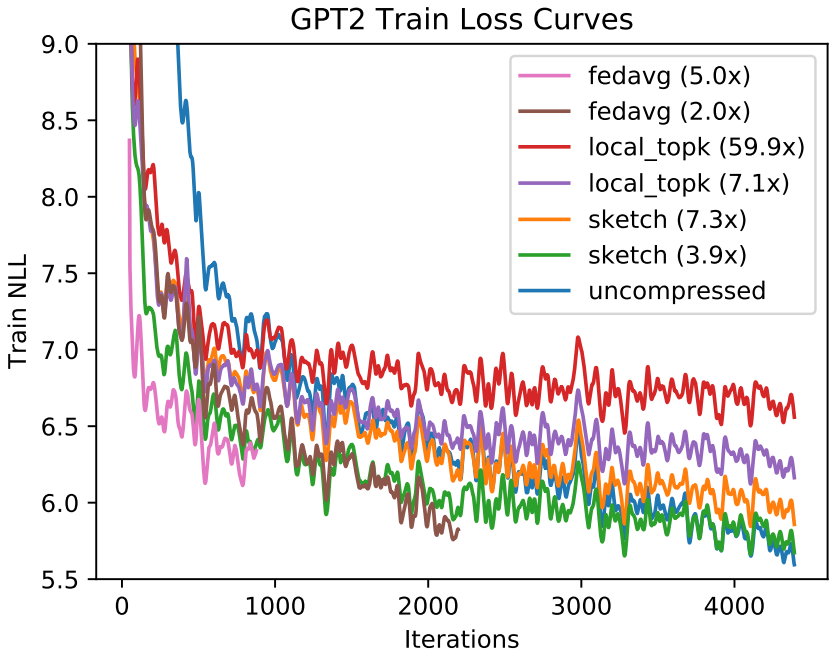


Figure 11: Training curves for the GPT2 runs reported in Table 1. Federated averaging runs end early since the only way federated averaging achieves compression is by carrying out fewer communication rounds, and in this task each client participates only once.

(a) Naive sliding window

(b) Adaptive sliding window

Figure 12: Approximating ℓ_2 norm on sliding windows.