
FetchSGD: Communication-Efficient Federated Learning with Sketching

Daniel Rothchild^{* 1} Ashwinee Panda^{* 1} Enayat Ullah² Nikita Ivkin³ Ion Stoica¹ Vladimir Braverman²
Joseph Gonzalez¹ Raman Arora²

Abstract

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

1. Introduction

Federated learning has recently emerged as an important setting for training machine learning models. In the federated setting, training data is distributed across a large number of edge devices, such as consumer smartphones, personal computers, or smart home devices. These devices have data that is useful for training a variety of models – for text prediction, speech modeling, facial recognition, document identification, and other tasks (Shi et al., 2016; Brisimi et al., 2018; Leroy et al., 2019; Tomlinson et al., 2009). However, data privacy, liability, or regulatory concerns may make it difficult to move this data to the cloud for training (EU,

2018). Even without these concerns, training machine learning models in the cloud can be expensive, and an effective way to train the same models on the edge has the potential to eliminate this expense.

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

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

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

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

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

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

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

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

2. Related Work

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

2.1. FedAvg

FedAvg reduces the total number of bytes transferred during training by carrying out multiple steps of stochastic gradient descent (SGD) locally before sending the aggregate model update back to the aggregator. This technique,

often referred to as local/parallel SGD, has been studied since the early days of distributed model training in the data center (Dean et al., 2012), and is referred to as FedAvg when applied to federated learning (McMahan et al., 2016). FedAvg has been successfully deployed in a number of domains (Hard et al., 2018; Li et al., 2019), and is the most commonly used optimization algorithm in the federated setting (Yang et al., 2018). In FedAvg, every participating client first downloads and trains the global model on their local dataset for a number of epochs using SGD. The clients upload the difference between their initial and final model to the parameter server, which averages the local updates weighted according to the magnitude of the corresponding local dataset.

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

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

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

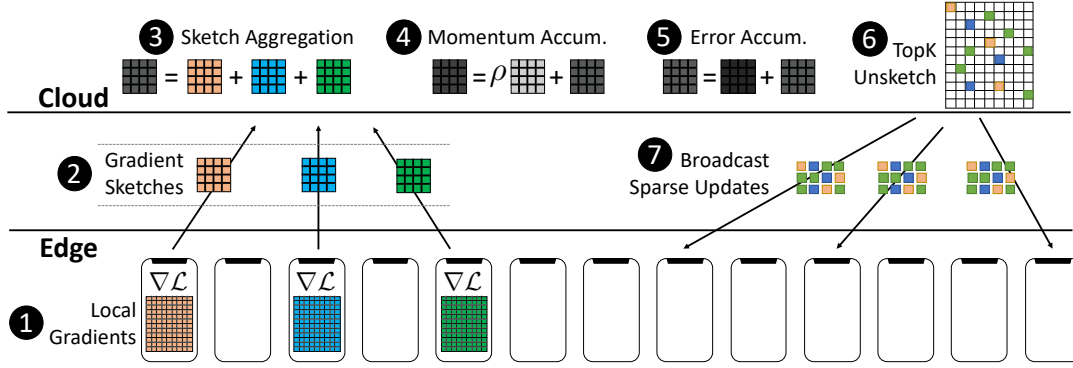


Figure 1. **Algorithm Overview.** The FetchSGD algorithm (1) computes gradients locally, and then send sketches (2) of the gradients to the cloud. In the cloud, gradient sketches are aggregated (3), and then (4) momentum and (5) error accumulation are applied to the sketch. The approximate top-k values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

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

2.2. Gradient Compression

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

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

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

2.3. Optimization with Sketching

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

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

3. FetchSGD

3.1. Federated Learning Setup

Consider a federated learning scenario with C clients. Let \mathcal{Z} be the data domain and let $\{\mathcal{P}_i\}_{i=1}^C$ be C possibly unrelated probability distributions over \mathcal{Z} . For supervised learning, $\mathcal{Z} = \mathcal{X} \times \mathcal{Y}$, where \mathcal{X} is the feature space and \mathcal{Y} is the label space; for unsupervised learning, $\mathcal{Z} = \mathcal{X}$ is the feature space. The i^{th} client has D_i samples drawn i.i.d. from the \mathcal{P}_i . Let \mathcal{W} be the hypothesis class parametrized by d dimensional vectors. Let $\mathcal{L} : \mathcal{W} \times \mathcal{Z} \rightarrow \mathbb{R}$ be a loss function. The goal is to minimize the weighted average $\hat{\mathbb{E}}$ of client risks:

$$f(\mathbf{w}) = \hat{\mathbb{E}} f_i(\mathbf{w}) = \frac{1}{\sum_{i=1}^C D_i} \sum_{i=1}^C D_i \mathbb{E}_{z \sim \mathcal{P}_i} \mathcal{L}(\mathbf{w}, z) \quad (1)$$

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

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

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

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

3.2. Algorithm

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

A Count Sketch is a randomized data structure that can compress a vector by randomly projecting it several times to lower dimensional spaces, such that high-magnitude elements can later be approximately recovered. We provide more details on the Count Sketch in Appendix C, but here

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

we treat it simply as a compression operator $\mathcal{S}(\cdot)$, with the special property that it is linear:

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

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

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

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

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

Briefly, $\mathcal{U}(\cdot)$ approximately “undoes” the projections computed by $\mathcal{S}(\cdot)$, and then uses these reconstructions to estimate the original vector. See Appendix C for more details.

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

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

where η is the learning rate and $\Delta^t \in \mathbb{R}^d$ is k -sparse.

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

Viewed another way, because $\mathcal{S}(\cdot)$ is linear, and because error accumulation consists only of linear operations, carrying out error accumulation on the server within S_e is equivalent

to carrying out **error accumulation on each client**, and uploading sketches of the result to the server. (Computing the model update from the accumulated error is not linear, but only the server does this, whether the error is accumulated on the clients or on the server.) Taking this a step further, we note that **momentum** also consists of only linear operations, and so momentum can be equivalently carried out on the clients or on the server. Extending the above equations with momentum yields

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

FetchSGD is presented in full in Algorithm 1.

Algorithm 1 FetchSGD

Input: number of model weights to update each round k
Input: learning rate η
Input: number of timesteps T
Input: momentum parameter ρ , local batch size ℓ
Input: Number of clients selected per round W
Input: Sketching and unsketching functions \mathcal{S}, \mathcal{U}
1: Initialize S_u^0 and S_e^0 to zero sketches
2: Initialize w^0 using the same random seed on the clients and aggregator
3: **for** $t = 1, 2, \dots, T$ **do**
4: Randomly select W clients c_1, \dots, c_W
5: **loop** {In parallel on clients $\{c_i\}_{i=1}^W$ }
6: Download (possibly sparse) new model weights $w^t - w^0$
7: Compute stochastic gradient g_i^t on batch B_i of size ℓ :
 $g_i^t = \frac{1}{\ell} \sum_{j=1}^{\ell} \nabla_w \mathcal{L}(w^t, z_j)$
8: Sketch g_i^t : $S_i^t = \mathcal{S}(g_i^t)$ and send it to the Aggregator
9: **end loop**
10: Aggregate sketches $S^t = \frac{1}{W} \sum_{i=1}^W S_i^t$
11: Momentum: $S_u^t = \rho S_u^{t-1} + S^t$
12: Error feedback: $S_e^t = \eta S_u^t + S_e^{t-1}$
13: Unsketch: $\Delta^t = \text{Top-k}(\mathcal{U}(S_e^t))$
14: Error accumulation: $S_e^{t+1} = S_e^t - S(\Delta^t)$
15: Update $w^{t+1} = w^t - \Delta^t$
16: **end for**
Output: $\{w^t\}_{t=1}^T$

4. Theory

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

4.1. Scenario 1: Contraction Holds

To show that compressed SGD converges when using some biased gradient compression operator $\mathcal{C}(\cdot)$, existing methods (Karimireddy et al., 2019b; Zheng et al., 2019; Ivkin et al., 2019b) appeal to Stich et al. (2018), who show that compressed SGD converges when \mathcal{C} is a τ -contraction:

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

Ivkin et al. (2019b) show that it is possible to satisfy this contraction property using Count Sketches to compress gradients. However, their compression method includes a **second round of communication**: if there are no high-magnitude elements in e^t , as computed from $\mathcal{S}(e^t)$, the server can query clients for random entries of e^t . On the other hand, FetchSGD never computes the e_i^t , or e^t , so this second round of communication is not possible, and the analysis of Ivkin et al. (2019b) does not apply. In this section, we assume that the updates have heavy hitters, which ensures that the contraction property holds along the optimization path.

Assumption 1 (Scenario 1). *Let $\{w_t\}_{t=1}^T$ be the sequence of models generated by FetchSGD. Fixing this model sequence, let $\{u^t\}_{t=1}^T$ and $\{e^t\}_{t=1}^T$ be the **momentum** and **error accumulation** vectors generated using this model sequence, had we not used sketching for gradient compression (i.e. if \mathcal{S} and \mathcal{U} are identity maps). There exists a constant $0 < \tau < 1$ such that for any $t \in [T]$, the quantity $q^t := \eta(\rho u^{t-1} + g^{t-1}) + e^{t-1}$ has at least one coordinate i s.t. $(q_i^t)^2 \geq \tau \|q^t\|^2$.*

Theorem 1 (Scenario 1). *Let f be an L -smooth² non-convex function and let the norm of stochastic gradients of f be upper bounded by G . Under Assumption 1, FetchSGD, with step size $\eta = \frac{1-\rho}{2L\sqrt{T}}$, in T iterations, returns $\{w^t\}_{t=1}^T$, such that, with probability at least $1 - \delta$ over the sketching randomness:*

1. $\min_{t=1 \dots T} \mathbb{E} \|\nabla f(w^t)\|^2 \leq \frac{4L(f(w^0) - f^*) + G^2}{\sqrt{T}} + \frac{2(1+\tau)^2 G^2}{(1-\tau)\tau^2 T}.$
2. *The sketch uploaded from each participating client to the parameter server is $\mathcal{O}(\log(dT/\delta)/\tau)$ bytes per round.*

The expectation in part 1 of the theorem is over the randomness of sampling minibatches. For large T , the first term dominates, so the convergence rate in Theorem 1 matches that of uncompressed SGD.

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,

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

momentum, and error accumulation vectors, and it is not immediately obvious that we should expect it to hold. To remedy this, the next section analyzes FetchSGD under a simpler assumption that involves only the gradients. Note that this is still an assumption on the algorithmic path, but it presents a clearer understanding.

4.2. Scenario 2: Sliding Window Heavy Hitters

Gradients taken along the optimization path have been observed to contain heavy coordinates (Shi et al., 2019; Li et al., 2019). However, it would be overly optimistic to assume that all gradients contain heavy coordinates, since this might not be the case in some flat regions of parameter space. Instead, we introduce a much milder assumption: namely that there exist heavy coordinates in a sliding sum of gradient vectors:

Definition 1. $[(I, \tau)\text{-sliding heavy}]^3$

A stochastic process $\{g^t\}_{t \in \mathbb{N}}$ is (I, τ) -sliding heavy if with probability at least $1 - \delta$, at every iteration t , the gradient vector g^t can be decomposed as $g^t = g_N^t + g_S^t$, where g_S^t is “signal” and g_N^t is “noise” with the following properties:

1. **[Signal]** For every non-zero coordinate j of vector g_S^t , $\exists t_1, t_2$ with $t_1 \leq t \leq t_2$, $t_2 - t_1 \leq I$ s.t. $|\sum_{t=t_1}^{t_2} g_j^t| > \tau \|\sum_{t=t_1}^{t_2} g^t\|$.
2. **[Noise]** g_N^t is mean zero, symmetric and when normalized by its norm, its second moment bounded as $\mathbb{E} \frac{\|g_N^t\|^2}{\|g^t\|^2} \leq \beta$.

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

Under this definition, we can use Count Sketches to capture the signal, since Count Sketches can approximate heavy hitters. Because the signal is spread over sliding windows of size I , we need a sliding window error accumulation

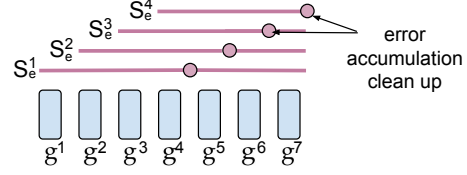


Figure 2. Sliding window error accumulation

scheme to ensure that we capture whatever signal is present. Vanilla error accumulation is not sufficient to show convergence, since vanilla error accumulation sums up all prior gradients, so signal that is present only in a sum of I consecutive gradients (but not in $I + 1$, or $I + 2$, etc.) will not be captured with vanilla error accumulation. Instead, we can use a sliding window error accumulation scheme, which can capture any signal that is spread over a sequence of at most I gradients. One simple way to accomplish this is to maintain I error accumulation Count Sketches, as shown in Figure 2 for $I = 4$. Each sketch accumulates new gradients every iteration, and beginning at offset iterations, each sketch is zeroed out every I iterations before continuing to accumulate gradients (this happens after line 15 of Algorithm 1). Under this scheme, at every iteration there is a sketch available that contains the sketched sum of the prior I' gradients, for all $I' \leq I$. We prove convergence in Theorem 2 when using this sort of sliding window error accumulation scheme.

In practice, it is too expensive to maintain I error accumulation sketches. Fortunately, this “sliding window” problem is well studied (Datar et al., 2002; Braverman and Ostrovsky, 2007; Braverman et al., 2014; 2015; 2018b;a), and it is possible to identify heavy hitters with only $\log(I)$ error accumulation sketches. Additional details on sliding window Count Sketch are in Appendix D. Although we use a sliding window error accumulation scheme to prove convergence, in all experiments we use a single error accumulation sketch, since we find that doing so still leads to good convergence.

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

Theorem 2 (Scenario 2). *Let f be an L -smooth non-convex function and let g_i denote stochastic gradients of f_i such that $\|g_i\|^2 \leq G^2$. Under Assumption 2, FetchSGD, using a sketch size $\Theta\left(\frac{\log(dT/\delta)}{\tau^2}\right)$, with step size $\eta = \frac{1}{G\sqrt{LT}^{2/3}}$ and $\rho = 0$ (no momentum), in T iterations, with probability at least $1 - 2\delta$, returns $\{w^t\}_{t=1}^T$ such that*

1. $\min_{t=1 \dots T} \mathbb{E} \|\nabla f(w^t)\|^2 \leq \frac{G\sqrt{L}(f(w^0) - f^*) + 2(2-\tau)}{T^{1/3}} + \frac{G\sqrt{L}}{T^{2/3}} + \frac{2l^2}{T^{4/3}}$
2. *The sketch uploaded from each participating client to the parameter server is $\Theta\left(\frac{\log(dT/\delta)}{\tau^2}\right)$ bytes per round.*

As in Theorem 1, the expectation in part 1 of the theorem is over the randomness of sampling minibatches.

³Technically, this definition is also parametrized by δ and β . However, in the interest of brevity, we use the simpler term “ (I, τ) -sliding heavy” throughout the manuscript. Note that δ in Theorem 2 refers to the same δ as in Definition 1.

Remarks:

1. These guarantees are for the non-i.i.d. setting – i.e. f is the average risk with respect to potentially unrelated distributions (see eqn. 2).
2. The convergence rates bound the objective gradient norm rather than the objective itself.
3. The convergence rate in Theorem 1 matches that of uncompressed SGD, while the rate in Theorem 2 is worse.
4. 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).

5. Evaluation

We implement and compare FetchSGD, gradient sparsification (local top- k), and FedAvg using PyTorch (Paszke et al., 2019).⁴ In contrast to our theoretical assumptions, we use **neural networks with ReLU activations**, whose loss surfaces are not L -smooth. In addition, although Theorem 2 uses a sliding window Count Sketch for error accumulation, **in practice we use a vanilla Count Sketch**. Lastly, we use **non-zero momentum**, which Theorem 1 allows but Theorem 2 does not. We also make two changes to Algorithm 1. For all methods, we employ **momentum factor masking** (Lin et al., 2017). And on line 14 of Algorithm 1, we zero out the nonzero coordinates of $\mathcal{S}(\Delta^t)$ in S_ϕ^t instead of subtracting $\mathcal{S}(\Delta^t)$; empirically, doing so stabilizes the optimization.

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

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

to a more representative sample of the full data distribution.

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

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

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

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

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

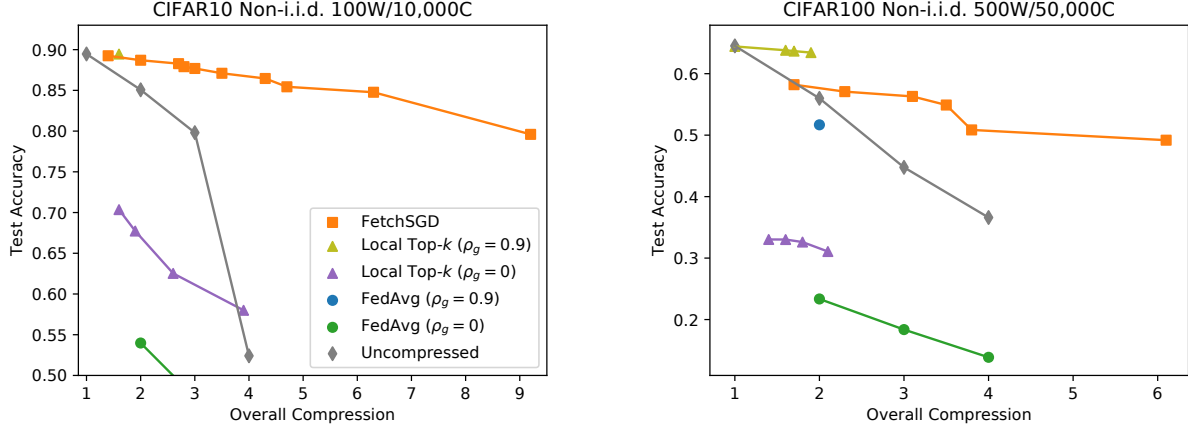


Figure 3. Test accuracy achieved on CIFAR10 (left) and CIFAR100 (right). “Uncompressed” refers to runs that attain compression by simply running for fewer epochs. FetchSGD outperforms all methods, especially at higher compression. Many FedAvg and local top- k runs are excluded from the plot because they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

FetchSGD incorporates momentum seamlessly due to the linearity of our compression operator (see Section 3.2); we use a momentum parameter of 0.9 in all experiments.

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

5.1. CIFAR (ResNet9)

CIFAR10 and CIFAR100 (Krizhevsky et al., 2009) are image classification datasets with 60,000 32×32 px color images distributed evenly over 10 and 100 classes respectively (50,000/10,000 train/test split). They are benchmark computer vision datasets, and although they lack a natural non-i.i.d. partitioning, we artificially create one by giving each client images from only a single class. For CIFAR10 (CIFAR100) we use 10,000 (50,000) clients, yielding 5 (1) images per client. Our 7M-parameter model architecture, data preprocessing, and most hyperparameters follow Page (2019), with details in Appendix A.1. We report accuracy on the test datasets.

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

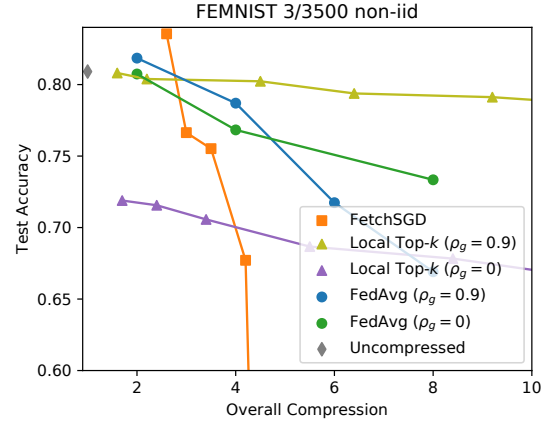


Figure 4. Test accuracy on FEMNIST. The dataset is not very non-i.i.d., and has relatively large local datasets, but FetchSGD is still competitive with FedAvg and local top- k for lower compression.

from many workers, each with very different data, leads to a nearly dense model update each round.

5.2. FEMNIST (ResNet101)

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

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 single person. Experimental details, including our 40M-parameter model architecture, can be found Appendix A.2. We report final accuracies on the validation dataset. The baseline run trains for a single epoch (*i.e.*, each client participates once).

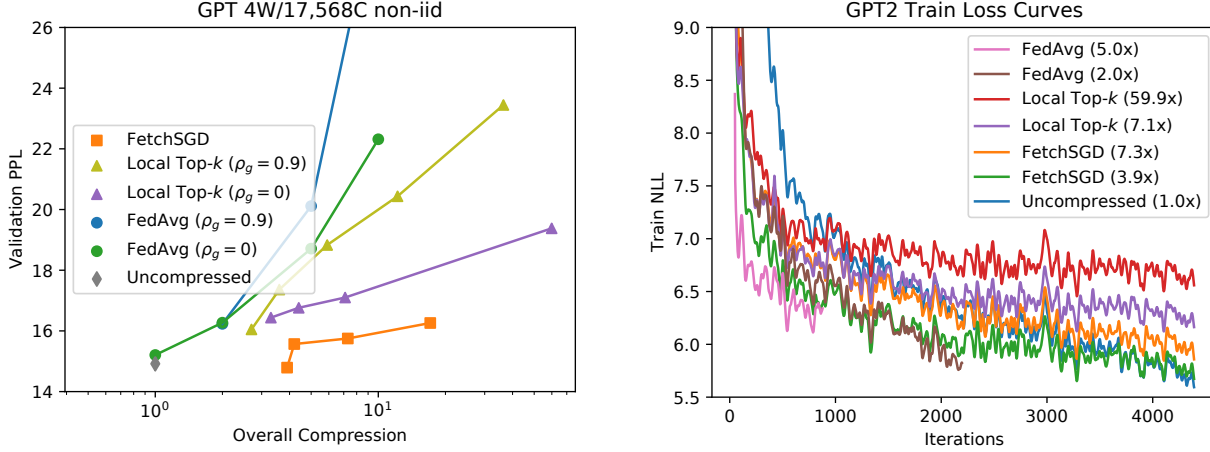


Figure 5. Left: Validation perplexity achieved by finetuning GPT2-small on PersonaChat. FetchSGD achieves 3.9 \times compression without loss in accuracy over uncompressed SGD, and it consistently achieves lower perplexity than FedAvg and top- k runs with similar compression. Right: Training loss curves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

FEMNIST was introduced as a benchmark dataset for FedAvg, and it has relatively large local datasets (~ 200 images per client). The clients are split according to the person who wrote the character, yielding a data distribution closer to i.i.d. than our per-class splits of CIFAR10. To maintain a reasonable overall batch size, only three clients participate each round, reducing the need for a linear compression operator. Despite this, FetchSGD performs competitively with both FedAvg and local top- k for some compression values, as shown in Figure 4.

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

5.3. PersonaChat (GPT2)

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

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

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

6. Discussion

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

Acknowledgements

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

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

References

- Dan Alistarh, Demjan Grubic, Jerry Li, Ryota Tomioka, and Milan Vojnovic. Qsgd: Communication-efficient sgd via gradient quantization and encoding. In *Advances in Neural Information Processing Systems*, pages 1709–1720, 2017.
- Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. *Journal of Computer and system sciences*, 58(1):137–147, 1999.
- Eugene Bagdasaryan, Andreas Veit, Yiqing Hua, Deborah Estrin, and Vitaly Shmatikov. How to backdoor federated learning, 2018.
- Jeremy Bernstein, Yu-Xiang Wang, Kamyar Azizzadenesheli, and Anima Anandkumar. signsgd: Compressed optimisation for non-convex problems. *arXiv preprint arXiv:1802.04434*, 2018.
- Arjun Nitin Bhagoji, Supriyo Chakraborty, Prateek Mittal, and Seraphin Calo. Analyzing federated learning through an adversarial lens. *arXiv preprint arXiv:1811.12470*, 2018.
- Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for federated learning on user-held data. *arXiv preprint arXiv:1611.04482*, 2016.
- Vladimir Braverman and Rafail Ostrovsky. Smooth histograms for sliding windows. In *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS’07)*, pages 283–293. IEEE, 2007.
- Vladimir Braverman, Ran Gelles, and Rafail Ostrovsky. How to catch ℓ_2 -heavy-hitters on sliding windows. *Theoretical Computer Science*, 554:82–94, 2014.
- Vladimir Braverman, Rafail Ostrovsky, and Alan Roytman. Zero-one laws for sliding windows and universal sketches. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2015)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2015.
- Vladimir Braverman, Stephen R Chestnut, Nikita Ivkin, Jelani Nelson, Zhengyu Wang, and David P Woodruff. Bptree: an ℓ_2 heavy hitters algorithm using constant memory. In *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 361–376, 2017.
- Vladimir Braverman, Petros Drineas, Cameron Musco, Christopher Musco, Jalaj Upadhyay, David P Woodruff, and Samson Zhou. Near optimal linear algebra in the online and sliding window models. *arXiv preprint arXiv:1805.03765*, 2018a.
- Vladimir Braverman, Elena Grigorescu, Harry Lang, David P Woodruff, and Samson Zhou. Nearly optimal distinct elements and heavy hitters on sliding windows. *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, 2018b.
- Theodora S Brisimi, Ruidi Chen, Theofanie Mela, Alex Olshchovsky, Ioannis Ch Paschalidis, and Wei Shi. Federated learning of predictive models from federated electronic health records. *International journal of medical informatics*, 112:59–67, 2018.
- Sebastian Caldas, Sai Meher Karthik Duddu, Peter Wu, Tian Li, Jakub Konecny, H. Brendan McMahan, Virginia Smith, and Ameet Talwalkar. Leaf: A benchmark for federated settings, 2018.
- Moses Charikar, Kevin Chen, and Martin Farach-Colton. Finding frequent items in data streams. In *International Colloquium on Automata, Languages, and Programming*, pages 693–703. Springer, 2002.
- G. Cohen, S. Afshar, J. Tapson, and A. van Schaik. Emnist: Extending mnist to handwritten letters. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 2921–2926, May 2017. doi: 10.1109/IJCNN.2017.7966217.
- Mayur Datar, Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Maintaining stream statistics over sliding windows. *SIAM journal on computing*, 31(6):1794–1813, 2002.
- Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Marc’auelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, et al. Large scale distributed deep networks. In *Advances in neural information processing systems*, pages 1223–1231, 2012.

- EU. 2018 reform of eu data protection rules, 2018. URL <https://tinyurl.com/ydaltt5g>.
- Robin C. Geyer, Tassilo Klein, and Moin Nabi. Differentially private federated learning: A client level perspective, 2017.
- Oana Goga and Renata Teixeira. Speed measurements of residential internet access. In *International Conference on Passive and Active Network Measurement*, pages 168–178. Springer, 2012.
- Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large mini-batch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.
- Andrew Hard, Kanishka Rao, Rajiv Mathews, Swaroop Ramaswamy, Francoise Beaufays, Sean Augenstein, Hubert Eichner, Chloé Kiddon, and Daniel Ramage. Federated learning for mobile keyboard prediction. *arXiv preprint arXiv:1811.03604*, 2018.
- Stephen Hardy, Wilko Henecka, Hamish Ivey-Law, Richard Nock, Giorgio Patrini, Guillaume Smith, and Brian Thorne. Private federated learning on vertically partitioned data via entity resolution and additively homomorphic encryption. *arXiv preprint arXiv:1711.10677*, 2017.
- Nikita Ivkin, Zaoxing Liu, Lin F Yang, Srinivas Suresh Kumar, Gerard Lemson, Mark Neyrinck, Alexander S Szalay, Vladimir Braverman, and Tamas Budavari. Scalable streaming tools for analyzing n-body simulations: Finding halos and investigating excursion sets in one pass. *Astronomy and computing*, 23:166–179, 2018.
- Nikita Ivkin, Ran Ben Basat, Zaoxing Liu, Gil Einziger, Roy Friedman, and Vladimir Braverman. I know what you did last summer: Network monitoring using interval queries. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 3(3):1–28, 2019a.
- Nikita Ivkin, Daniel Rothchild, Enayat Ullah, Vladimir Braverman, Ion Stoica, and Raman Arora. Communication-efficient distributed sgd with sketching. In *Advances in Neural Information Processing Systems*, pages 13144–13154, 2019b.
- Nikita Ivkin, Zhuolong Yu, Vladimir Braverman, and Xin Jin. Qpipe: Quantiles sketch fully in the data plane. In *Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies*, pages 285–291, 2019c.
- Jiawei Jiang, Fangcheng Fu, Tong Yang, and Bin Cui. Sketchml: Accelerating distributed machine learning with data sketches. In *Proceedings of the 2018 International Conference on Management of Data*, pages 1269–1284, 2018.
- Peter Kairouz, H. Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Keith Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, Rafael G. L. D’Oliveira, Salim El Rouayheb, David Evans, Josh Gardner, Zachary Garrett, Adrià Gascón, Badi Ghazi, Phillip B. Gibbons, Marco Gruteser, Zaid Harchaoui, Chaoyang He, Lie He, Zhouyuan Huo, Ben Hutchinson, Justin Hsu, Martin Jaggi, Tara Javidi, Gauri Joshi, Mikhail Khodak, Jakub Konecny, Aleksandra Korolova, Farinaz Koushanfar, Sanmi Koyejo, Tancrède Lepoint, Yang Liu, Prateek Mittal, Mehryar Mohri, Richard Nock, Ayfer Özgür, Rasmus Pagh, Mariana Raykova, Hang Qi, Daniel Ramage, Ramesh Raskar, Dawn Song, Weikang Song, Sebastian U. Stich, Ziteng Sun, Ananda Theertha Suresh, Florian Tramèr, Praneeth Vepakomma, Jianyu Wang, Li Xiong, Zheng Xu, Qiang Yang, Felix X. Yu, Han Yu, and Sen Zhao. Advances and open problems in federated learning, 2019.
- Sai Praneeth Karimireddy, Satyen Kale, Mehryar Mohri, Sashank J. Reddi, Sebastian U. Stich, and Ananda Theertha Suresh. Scaffold: Stochastic controlled averaging for on-device federated learning, 2019a.
- Sai Praneeth Karimireddy, Quentin Rebjock, Sebastian U Stich, and Martin Jaggi. Error feedback fixes signsgd and other gradient compression schemes. *arXiv preprint arXiv:1901.09847*, 2019b.
- Jakub Konecny, H. Brendan McMahan, Felix X. Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. Federated learning: Strategies for improving communication efficiency, 2016.
- Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. *Master’s thesis, Department of Computer Science, University of Toronto*, 2009.
- Kyunghan Lee, Joohyun Lee, Yung Yi, Injong Rhee, and Song Chong. Mobile data offloading: How much can wifi deliver? In *Proceedings of the 6th International Conference*, pages 1–12, 2010.
- David Leroy, Alice Coucke, Thibaut Lavril, Thibault Giselbrecht, and Joseph Dureau. Federated learning for keyword spotting. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6341–6345. IEEE, 2019.

- He Li, Kaoru Ota, and Mianxiong Dong. Learning iot in edge: Deep learning for the internet of things with edge computing. *IEEE network*, 32(1):96–101, 2018.
- Tian Li, Zaoxing Liu, Vyas Sekar, and Virginia Smith. Privacy for free: Communication-efficient learning with differential privacy using sketches. *arXiv preprint arXiv:1911.00972*, 2019.
- Yujun Lin, Song Han, Huizi Mao, Yu Wang, and William J Dally. Deep gradient compression: Reducing the communication bandwidth for distributed training. *arXiv preprint arXiv:1712.01887*, 2017.
- Zaoxing Liu, Nikita Ivkin, Lin Yang, Mark Neyrinck, Gerard Lemson, Alexander Szalay, Vladimir Braverman, Tamas Budavari, Randal Burns, and Xin Wang. Streaming algorithms for halo finders. In *2015 IEEE 11th International Conference on e-Science*, pages 342–351. IEEE, 2015.
- H Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, et al. Communication-efficient learning of deep networks from decentralized data. *arXiv preprint arXiv:1602.05629*, 2016.
- Jayadev Misra and David Gries. Finding repeated elements. *Science of computer programming*, 2(2):143–152, 1982.
- Lev Muchnik, Sen Pei, Lucas C Parra, Saulo DS Reis, José S Andrade Jr, Shlomo Havlin, and Hernán A Makse. Origins of power-law degree distribution in the heterogeneity of human activity in social networks. *Scientific reports*, 3(1):1–8, 2013.
- Shanmugavelayutham Muthukrishnan et al. Data streams: Algorithms and applications. *Foundations and Trends® in Theoretical Computer Science*, 1(2):117–236, 2005.
- David Page. How to train your resnet, Nov 2019. URL <https://myrtle.ai/how-to-train-your-resnet/>.
- Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlché Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems* 32, pages 8024–8035. Curran Associates, Inc., 2019.
- Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8):9, 2019.
- Anit Kumar Sahu, Tian Li, Maziar Sanjabi, Manzil Zaheer, Ameet Talwalkar, and Virginia Smith. On the convergence of federated optimization in heterogeneous networks. *arXiv preprint arXiv:1812.06127*, 2018.
- Shaohuai Shi, Xiaowen Chu, Ka Chun Cheung, and Simon See. Understanding top-k sparsification in distributed deep learning. *arXiv preprint arXiv:1911.08772*, 2019.
- Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. Edge computing: Vision and challenges. *IEEE internet of things journal*, 3(5):637–646, 2016.
- Ryan Spring, Anastasios Kyrillidis, Vijai Mohan, and Anshumali Shrivastava. Compressing gradient optimizers via count-sketches. *arXiv preprint arXiv:1902.00179*, 2019.
- Sebastian U Stich, Jean-Baptiste Cordonnier, and Martin Jaggi. Sparsified sgd with memory. In *Advances in Neural Information Processing Systems*, pages 4447–4458, 2018.
- Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147, 2013.
- Mark Tomlinson, Wesley Solomon, Yages Singh, Tanya Doherty, Mickey Chopra, Petrida Ijumba, Alexander C Tsai, and Debra Jackson. The use of mobile phones as a data collection tool: a report from a household survey in south africa. *BMC medical informatics and decision making*, 9(1):51, 2009.
- Jianyu Wang and Gauri Joshi. Cooperative sgd: A unified framework for the design and analysis of communication-efficient sgd algorithms, 2018.
- Shiqiang Wang, Tiffany Tuor, Theodoros Salonidis, Kin K Leung, Christian Makaya, Ting He, and Kevin Chan. Adaptive federated learning in resource constrained edge computing systems. *IEEE Journal on Selected Areas in Communications*, 37(6):1205–1221, 2019.
- Jianqiao Wangni, Jiale Wang, Ji Liu, and Tong Zhang. Gradient sparsification for communication-efficient distributed optimization. In *Advances in Neural Information Processing Systems*, pages 1299–1309, 2018.
- Thomas Wolf. How to build a state-of-the-art conversational ai with transfer learning, May 2019. URL <https://tinyurl.com/ryehjbt>.
- Thomas Wolf, L Debut, V Sanh, J Chaumond, C Delangue, A Moi, P Cistac, T Rault, R Louf, M Funtowicz, et al. Huggingface’s transformers: State-of-the-art natural language processing. *ArXiv, abs/1910.03771*, 2019.

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

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

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

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

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