Adaptive Federated Optimization

Sashank J. Reddi* Google Research sashank@google.com
Zachary Charles* Google Research zachcharles@google.com
Manzil Zaheer Google Research manzilzaheer@google.com
Zachary Garrett Google Research zachgarrett@google.com
Keith Rush Google Research krush@google.com
Jakub Konečný Google Research konkey@google.com
Sanjiv Kumar Google Research sanjivk@google.com
H. Brendan McMahan Google Research mcmahan@google.com

Abstract

Federated learning is a distributed machine learning paradigm in which a large number of clients coordinate with a central server to learn a model without sharing their own training data. Due to the heterogeneity of the client datasets, standard federated optimization methods such as Federated Averaging (FEDAVG) are often difficult to tune and exhibit unfavorable convergence behavior. In non-federated settings, adaptive optimization methods have had notable success in combating such issues. In this work, we propose federated versions of adaptive optimizers, including ADAGRAD, ADAM, and YOGI, and analyze their convergence in the presence of heterogeneous data for general nonconvex settings. Our results highlight the interplay between client heterogeneity and communication efficiency. We also perform extensive experiments on these methods and show that the use of adaptive optimizers can significantly improve the performance of federated learning.

heterogeneity is the main motivation.

1 Introduction

Federated learning (FL) is a machine learning paradigm in which multiple clients (such as edge devices, or separate organizations) cooperate to learn a model, under the orchestration of a central server (McMahan et al., 2017). The core tenet of FL is that raw client data is never shared with the server or among distinct clients. While valuable from a privacy perspective, this distinguishes FL from traditional distributed optimization, as the data restrictions require FL to contend with *heterogeneous data*. For a more in-depth discussion of federated learning and the challenges therein, we defer to Kairouz et al. (2019) and Li et al. (2019a).

The standard approach of using *mini-batch* SGD is unsuitable in many FL settings as it can incur high communication costs. To this end, many optimization methods for FL utilize *local client updates*, in which clients update their local models multiple times before communicating to synchronize the client models. This can greatly reduce the amount of communication required to train a model to a given accuracy; one particularly popular local optimization method for federated learning is FEDAVG (McMahan et al., 2017). In each round of FEDAVG, a small fraction of the clients locally perform some number of epochs of SGD in parallel. The clients then communicate their model updates to the server, which averages them to compute a new global model.

reduce communication load

While FEDAVG has seen remarkable success in FL, several recent works have highlighted its drawbacks (Karimireddy et al., 2019; Hsu et al., 2019). In this paper, we focus on two issues: (a) *client drift* and (b) lack of *adaptive learning rates* during optimization. In heterogeneous settings, the multiple local SGD epochs can cause clients to drift away from a globally optimal model. For instance, the extreme case where each client *exactly* minimizes the loss over its local data using SGD and the server averages the models is equivalent to *one-shot averaging*, which is known to fail in heterogeneous settings. Moreover, FEDAVG, which is similar in spirit to SGD, is typically unsuitable for settings which exhibit heavy-tail stochastic gradient noise distributions during training (Zhang et al., 2019). Such settings necessitate the use of adaptive learning rates, which incorporate knowledge of the the past iterations to perform more informed

also talked about adding η_g i.e. global step-size in FedAvg.

*Authors contributed equally to this work.

gradient-based optimization. However, due to the decentralized nature of FL, incorporating adaptive learning rates into FEDAVG is challenging.

In this paper, we use a simple approach to tackle both of the aforementioned issues. We view FEDAVG through the lens of a general optimization framework in which (1) clients perform multiple epochs of model updates using some *client optimizer*, which minimizes the loss on its local data, (2) the server updates the global model using a gradient-based *server optimizer* on the average of the *model updates* from the clients in order to minimize the loss across clients. As we shall see later, FEDAVG is an instance of this framework where SGD is used as both client and server optimizer, and the server learning rate is 1. This viewpoint provides a natural way to decouple client and server learning rates in FEDAVG and yields a generalization of FEDAVG which can utilize different learning rate schedules on the client and server, thereby providing a means to control client drift.

theoretical guarantees on decoupling given in "SCAFFOLD".

This generalization of FEDAVG is particularly appealing due to its simplicity and its ability to seamlessly incorporate adaptivity in the optimization process by using adaptive optimizers as client or server optimizers. Adaptive methods, such as ADAGRAD, ADAM and YOGI, are popular in the machine learning community because they can reduce the need to tune the learning rates and often exhibit strong empirical performance especially in presence of heavy-tail noise distributions of stochastic gradients (Reddi et al., 2019; McMahan and Streeter, 2010; Duchi et al., 2011; Zhang et al., 2019). Building upon this framework, we develop novel adaptive optimization techniques for FL by using per-coordinate adaptive methods as server optimizers, with an aim towards reducing the need to tune learning rates and the number of communication rounds required to reach a desired performance level.

perform good with heavy-tail noisy stochastic gradients.

Main Contributions In light of the above, we highlight the main contributions of our work.

nice concept of generalization.

- We study a general framework for optimization in the FL setting using the notions of server and client optimizers. This framework encapsulates and generalizes many prior optimization methods for FL, including FEDAVG.
- Based on this framework, we propose adaptive optimization techniques for FL and provide convergence analysis in general nonconvex settings. To our knowledge, these are the first adaptive optimizers for FL. We show an important interplay between the number of local steps and the amount of heterogeneity in the clients and quantify their effect on the convergence rates.
- We demonstrate strong empirical performance of the proposed adaptive optimizers on a diverse and representative set of six FL tasks, covering both image and text data. These experiments also show the importance of momentum and decreasing client learning rate schedules. We improve over baseline FEDAVG on all six tasks, with dramatic improvements on four of them. For example, we improve accuracy on a real-world next-word-prediction task from 9.5% accuracy to 22.2% accuracy. Improvements on a convex tag-prediction task are even larger. We provide an open-source implementation of our adaptive optimizers, as well as code to reproduce our empirical results[1].

Related Work FEDAVG was first introduced by McMahan et al. (2017), who demonstrated its significant performance in federated learning. Many variants have been since proposed to tackle issues such as stability and client drift. Examples include methods that add a regularization term in the client objectives towards the broadcast model (Li et al., 2018), and methods that use momentum on the server (Hsu et al., 2019). Karimireddy et al. (2019) use control variates to reduce the client drift. While effective in some cases, these require clients to have state that persists throughout the training process.

requires client to have state that persists.

When the client are homogeneous, FEDAVG reduces to local SGD (Zinkevich et al., 2010), which has been analyzed theoretically by many works (Stich, 2019; Yu et al., 2019; Wang and Joshi, 2018; Stich and Karimireddy, 2019). In order to analyze FEDAVG in heterogeneous settings, many works derive convergence rates that depend on the amount of heterogeneity (Li et al., 2018; Wang et al., 2019; Khaled et al., 2019; Li et al., 2019b). The control variates used by Karimireddy et al. (2019) allow them to derive convergence rates independent of the amount of heterogeneity. For a more in-depth comparison, we defer to Kairouz et al. (2019).

[1] https://github.com/tensorflow/federated/tree/master/tensorflow_federated/python/research/optimization

There is a wide array of literature on the theoretical and empirical benefits of adaptive optimizers in non-federated settings, both in convex settings (McMahan and Streeter, 2010; Duchi et al., 2011; Kingma and Ba, 2015) and non-convex setting (Zaheer et al., 2018; Li and Orabona, 2018; Ward et al., 2018; Wu et al., 2019). Zaheer et al. (2018) study convergence failures of adaptive algorithms in non-convex settings, and develop an adaptive optimizer, YOGI, designed to improve convergence over methods such as ADAM. The aforementioned work focuses exclusively on non-federated optimization. We give, to the best of our knowledge, the first analysis of adaptive optimizers for federated learning.

Notation For vectors $a, b \in \mathbb{R}^d$, we use $\sqrt{a}$ for element-wise square root, a^2 for element-wise square, a/b to denote element-wise division. For any vector $\theta_i \in \mathbb{R}^d$, either $\theta_{i,j}$ or $[\theta_i]_j$ are used to denote its j^{th} coordinate.

2 Federated Learning and FEDAVG

In federated learning, we often wish to solve

$$\arg\min_{x \in \mathbb{R}^d} f(x) = \frac{1}{m} \sum_{i=1}^{m} F_i(x), \tag{1}$$

where $F_i(x) = \mathbb{E}_{z \sim \mathcal{D}_i}[f_i(x, z)]$, is the loss function of the i^{th} client, $z \in \mathcal{Z}$, and $\mathcal{D}_i$ is the data distribution for the i^{th} client. For $i \neq j$, $\mathcal{D}_i$ and $\mathcal{D}_j$ may be very different. The functions F_i can be potentially nonconvex and thus, the overall function f is nonconvex. At each iteration, we assume access to an *unbiased* stochastic gradient $g_i(x) = \nabla F_i(x)$ corresponding to the parameters x. In addition, we make the following assumptions.

Assumption 1 (Lipschitz Gradient)**.** *The function F_i is L-smooth for all $i \in [m]$ i.e., $\|\nabla F_i(x) - \nabla F_i(y)\| \leq L\|x - y\|$, for all $x, y \in \mathbb{R}^d$.*

Assumption 2 (Bounded Variance)**.** *The function F_i have σ_l-bounded (local) variance i.e., $\mathbb{E}[\|\nabla[f_i(x, z)]_j - [\nabla F_i(x)]_j\|^2] = \sigma_{l,j}^2$ for all $x \in \mathbb{R}^d$, $j \in [d]$ and $i \in [m]$. Furthermore, we assume that the (global) variance is bounded, $\frac{1}{m}\sum_{i=1}^{m} \|\nabla[F_i(x)]_j - [\nabla f(x)]_j\|^2 \leq \sigma_{g,j}^2$ for all $x \in \mathbb{R}^d$ and $j \in [d]$.*

Assumption 3 (Bounded Gradients)**.** *The function $f_i(x, z)$ have G-bounded gradients i.e., for any $i \in [m]$, $x \in \mathbb{R}^d$ and $z \in \mathcal{Z}$ we have $|[\nabla f_i(x, z)]_j| \leq G$ for all $j \in [d]$.*

With a slight abuse of notation, we use σ_l^2 and σ_g^2 to denote $\sum_{j=1}^{d} \sigma_{l,j}^2$ and $\sum_{j=1}^{d} \sigma_{g,j}^2$ respectively. Assumptions 1 and 3 are fairly standard in nonconvex optimization literature (Reddi et al., 2016; Ward et al., 2018; Zaheer et al., 2018). In this paper, we make no further assumptions regarding the similarity of different clients datasets. Assumption 2 assumes a form of bounded variance, but between the client objective functions and the overall objective function. This assumption has been used throughout various works on federated optimization (Li et al., 2018; Wang et al., 2019). Intuitively, the parameter σ_g signifies the similarity of the functions of different clients. In particular, the case of $\sigma_g = 0$ corresponds to the $i.i.d$ setting.

$\sigma_g \equiv$ Similarity Control variable !

Algorithm 1 Simplified FEDAVG

Initialization: x_0
for $t = 0, \cdots, T-1$ **do**
  Sample subset $\mathcal{S}$ of clients
  $x_{i,0}^t = x_t$
  for each client $i \in \mathcal{S}$ **in parallel do**
    $x_i^t = \text{SGD}_K(x_t, \eta_l, F_i)$
  $x_{t+1} = \frac{1}{|\mathcal{S}|}\sum_{i \in \mathcal{S}} x_i^t$

Algorithm 2 Generalized FEDAVG

Initialization: x_0
for $t = 0, \cdots, T-1$ **do**
  Sample subset $\mathcal{S}$ of clients
  $x_{i,0}^t = x_t$
  for each client $i \in \mathcal{S}$ **in parallel do**
    for $k = 0, \cdots, K-1$ **do**
      Compute an unbiased estimate $g_{i,k}^t$ of $\nabla F_i(x_{i,k}^t)$
      $x_{i,k+1}^t = \text{CLIENTOPT}(x_{i,k}^t, g_{i,k}^t, \eta_l, t)$
    $\Delta_i^t = x_{i,K}^t - x_t$
  $\Delta_t = \frac{1}{|\mathcal{S}|}\sum_{i \in \mathcal{S}} \Delta_i^t$
  $x_{t+1} = \text{SERVEROPT}(x_t, -\Delta_t, \eta, t)$

One of the most common approaches to solving (1) in federated settings is FEDAVG (McMahan et al., 2017). At each round of this method, a subset of clients are selected (typically randomly) and the server broadcasts its global model to them. In parallel, these clients all run SGD on their own loss function, and send their model to the server. The server then updates its global model as the average of these local models. A simplified description of FEDAVG is given in Algorithm 1. We write $\text{SGD}_K(x_t, \eta_l, F_i)$ to denote K steps of SGD on gradients $\nabla f_i(x, z)$ for $z \sim \mathcal{D}_i$ with (local) learning rate η_l, starting from x_t .

To motivate the algorithms in the paper, we rewrite FEDAVG's update as

$$x_{t+1} = \frac{1}{|\mathcal{S}|}\sum_{i\in\mathcal{S}} x_{i,K}^t = x_t - \frac{1}{|\mathcal{S}|}\sum_{i\in\mathcal{S}} \left(x_t - x_{i,K}^t\right).$$

Let $\Delta_i^t := x_{i,K}^t - x_t$ and $\Delta_t = \frac{1}{|\mathcal{S}|}\sum_{i\in\mathcal{S}} \Delta_i^t$. Then the server update in FEDAVG is equivalent to applying SGD to the "pseudo-gradient" $-\Delta_t$ with learning rate $\eta = 1$. This expression makes it clear that other choices of learning rates (or schedules) become possible. Further, one could utilize an optimizer other than SGD on the clients, and use an alternative update rule on the server to apply the "pseudo-gradient". This family of algorithms is formalized as Algorithm 2.

Here, CLIENTOPT and SERVEROPT are *gradient-based* optimizers with learning rates η_l and η respectively. Intuitively, CLIENTOPT aims to optimize the objective based on their local data while SERVEROPT optimizes it from a global perspective. We note that Algorithm 2 naturally allows the use of adaptive optimizers (eg. ADAM, YOGI, etc.), as well as techniques such as server-side momentum (FEDAVGM) (Hsu et al., 2019). Furthermore, we allow η and η_l to depend on the round t in order to encompass learning rate schedules. In fact, as we shall see shortly, our theoretical and empirical analysis suggest that one has to decay the client learning rate to obtain good convergence.

While Algorithm 2 has intuitive benefits over FEDAVG, it also raises a fundamental question: *Can the negative of average model difference Δ_t be used as a pseudo-gradient in general server optimizer updates?* In this paper, we provide an affirmative answer to this question by establishing a theoretical basis for this generalization. More specifically, we develop principled adaptive optimizers for FL based on this framework and study their convergence behavior.

3 Adaptive Federated Optimization

In this section, we specialize Algorithm 2 to settings where SERVEROPT is an adaptive optimization method (one of ADAGRAD, YOGI or ADAM) and CLIENTOPT is SGD. Algorithm 3 and 4 provide pseudo-code for federated versions of ADAGRAD and (FEDYOGI, FEDADAM) respectively. For simplicity, Algorithm 4 is stated without momentum, which is typically used in ADAM and YOGI. The parameter τ in all the algorithms controls their *degree of adaptivity*, with smaller values of τ representing higher degrees of adaptivity. We note that the updates of the adaptive methods are invariant to fixed multiplicative changes to the client learning rate η_l for appropriately chosen τ; although, as we shall see shortly, we still require η_l to be small enough.

We provide convergence analysis of these algorithms in the general nonconvex setting, assuming *full participation*, i.e. $\mathcal{S} = [m]$. However, our analysis can be extended to more general federated learning settings with *limited participation*. Furthermore, non-uniform weighted average typically used in FEDAVG can also be incorporated into our analysis fairly easily (see McMahan et al. (2017)).

illustrated w/o momentum for simplicity.

Algorithm 3 FEDADAGRAD

Initialization: x_0, $\tau > 0$ and $v_{-1} \geq \tau^2$
for $t = 0, \cdots, T-1$ **do**
  Sample subset $\mathcal{S}$ of clients
  $x_{i,0}^t = x_t$
  for each client $i \in \mathcal{S}$ **in parallel do**
    for $k = 0, \cdots, K-1$ **do**
      Compute an unbiased estimate $g_{i,k}^t$ of $\nabla F_i(x_{i,k}^t)$
      $x_{i,k+1}^t = x_{i,k}^t - \eta_l g_{i,k}^t$
    $\Delta_i^t = x_{i,K}^t - x_t$
  $\Delta_t = \frac{1}{|\mathcal{S}|}\sum_{i\in\mathcal{S}} \Delta_i^t$
  $v_t = v_{t-1} + \Delta_t^2$
  $x_{t+1} = x_t + \eta \frac{\Delta_t}{\sqrt{v_t}+\tau}$

Algorithm 4 FEDYOGI (and FEDADAM)

Initialization: x_0, $v_{-1} \geq \tau^2$, decay $\beta_2 \in (0,1)$
for $t = 0, \cdots, T-1$ **do**
  Sample subset $\mathcal{S}$ of clients
  $x_{i,0}^t = x_t$
  for each client $i \in \mathcal{S}$ **in parallel do**
    for $k = 0, \cdots, K-1$ **do**
      Compute an unbiased estimate $g_{i,k}^t$ of $\nabla F_i(x_{i,k}^t)$
      $x_{i,k+1}^t = x_{i,k}^t - \eta_l g_{i,k}^t$
    $\Delta_i^t = x_{i,K}^t - x_{t-1}$
  $\Delta_t = \frac{1}{|\mathcal{S}|}\sum_{i\in\mathcal{S}} \Delta_i^t$
  $v_t = v_{t-1} - (1-\beta_2)\Delta_t^2 \operatorname{sign}(v_{t-1} - \Delta_t^2)$ **(FEDYOGI)**
  $v_t = \beta_2 v_{t-1} + (1-\beta_2)\Delta_t^2$ **(FEDADAM)**
  $x_{t+1} = x_t + \eta \frac{\Delta_t}{\sqrt{v_t}+\tau}$

Theorem 1. *Let Assumptions 1 to 3 hold, and L, G, σ_l, σ_g be defined therein. Suppose $\sigma^2 = \sigma_l^2 + 6K\sigma_g^2$. Consider the following two conditions for η_l. (Condition I): $\eta_l \leq \frac{1}{8LK}$ and (Condition II):*

$$\eta_l \leq \frac{1}{3K} \min\left\{ \frac{1}{T^{1/10}} \left[\frac{\tau^3}{L^2G^3}\right]^{1/5}, \frac{1}{T^{1/8}} \left[\frac{\tau^2}{L^3G\eta}\right]^{1/4} \right\}.$$

*Then the iterates of Algorithm 3 (*FEDADAGRAD*) satisfy:*

- *Under Condition I only,*

$$\min_{0\leq t\leq T-1} \mathbb{E}\|\nabla f(x_t)\|^2 \leq \mathcal{O}\left(\left[\frac{G}{\sqrt{T}} + \frac{\tau}{\eta_l KT}\right](\Psi + \Psi_{var}) \right).$$

- *When both Condition I & II are satisfied,*

$$\min_{0\leq t\leq T-1} \mathbb{E}\|\nabla f(x_t)\|^2 \leq \mathcal{O}\left(\left[\frac{G}{\sqrt{T}} + \frac{\tau}{\eta_l KT}\right](\Psi + \widetilde{\Psi}_{\text{var}}) \right).$$

Here, Ψ, Ψ_{var} and $\widetilde{\Psi}_{var}$ are defined as:

$$\Psi = \frac{f(x_0) - f(x^*)}{\eta} + \frac{5\eta_l^3 K^2 L^2 T}{2\tau}\sigma^2,$$
$$\Psi_{var} = \frac{d(\eta_l KG^2 + \tau\eta L)}{\tau}\left[1 + \log\left(1 + \frac{\eta_l^2 K^2 G^2 T}{\tau^2}\right)\right],$$
$$\widetilde{\Psi}_{\text{var}} = \frac{2\eta_l KG^2 + \tau\eta L}{\tau^2}\left[\frac{2\eta_l^2 KT}{m}\sigma_l^2 + 10\eta_l^4 K^3 L^2 T\sigma^2\right].$$

All proofs are relegated to the Appendix due to space constraints. When η_l satisfies the condition in the second part the above result, we obtain a convergence rate depending on $\min\{\Psi_{\text{var}}, \widetilde{\Psi}_{\text{var}}\}$. To obtain an explicit dependence on T and K, we simplify the above result for a specific choice of η, η_l and τ.

Corollary 1. *Suppose η_l is such that conditions in Theorem 1 are satisfied. Furthermore, suppose client learning rate $\eta_l = \Theta(1/(KL\sqrt{T}))$, server learning rate $\eta = \Theta(\sqrt{Km})$ and $\tau = G/L$, then for sufficiently large T, we have the following for Algorithm 3 (*FEDADAGRAD*): $\min_{0\leq t\leq T-1} \mathbb{E}\|\nabla f(x_t)\|^2$ is*

$$\mathcal{O}\left(\frac{f(x_0) - f(x^*)}{\sqrt{mKT}} + \frac{2\sigma^2 L}{G^2\sqrt{mKT}} + \frac{\sigma^2}{GKT} + \frac{\sigma^2 L\sqrt{m}}{G^2\sqrt{KT^{3/2}}} \right).$$

We briefly discuss the implications of our convergence analysis here. First, it is interesting to note that we effectively obtain a convergence rate of $\mathcal{O}(\frac{1}{\sqrt{mKT}})$ when the client learning rate is $\Theta(\frac{1}{\sqrt{T}})$. While this requires knowledge the number of iterations T beforehand, it is easy to generalize our analysis to the case where η_l is decayed at a rate of $\frac{1}{\sqrt{t}}$. Also, observe that one has to decay the client learning rate and not the server learning rate to obtain convergence. This is due to the fact that the *client drift* introduced by the local updates does not vanish as $T \to \infty$ when η_l is not decayed. As we shall see later in our empirical analysis, such a decay of learning rate almost always improves the performance. In practice, one can decay the local learning rate at a slower rate, although, this is suboptimal from the viewpoint of the above convergence rate. Also, observe the inverse relationship between the client learning rate η_l and global learning rate η in the analysis above.

η_l decay imp. for mitigating client drift.

It is also instructive to note the interplay between the number of local steps K and the convergence rate. From Corollary 1, it may appear like increasing K leads to faster convergence. However, σ has an implicit dependence on K and σ_g, which controls the heterogeneity of the clients. When σ_g is large, one has to limit the number of local SGD updates to control the client drift. In particular:

with large variance b/w gradients, local training is restricted.

Corollary 2. *Suppose the parameters for* FEDADAGRAD *are chosen as specified in Corollary 1. Then, for sufficiently large T and $K = \mathcal{O}(\sigma_l^2/\sigma_g^2)$, we have*

$$\min_{0\le t\le T-1} \mathbb{E}\|\nabla f(x_t)\|^2 = \mathcal{O}\left(\frac{f(x_0)-f(x^*)}{\sqrt{mKT}} + \frac{\sigma_l^2 L}{G^2\sqrt{mKT}}\right).$$

Conceptually, the ratio of the local variance to global variance determines the parameter setting of K. In the setting where clients are homogeneous i.e., $\sigma_g = 0$, it is clear that large K is beneficial. On the other hand, when σ_g is large, it is important to limit K appropriately. For instance, in the extreme case where $K = \Theta(T)$, the algorithm may not even converge. Thus, the performance of the algorithm crucially relies on K and σ_g.

Analysis of FEDADAM Next, we provide the convergence analysis of FEDADAM. The proof of FEDYOGI is very similar and hence, we omit the details of FEDYOGI's analysis in the paper.

Theorem 2. *Let Assumptions 1 to 3 hold, and L, G, σ_l, σ_g be defined therein. Suppose $\sigma^2 = \sigma_l^2 + 6K\sigma_g^2$. Suppose the local learning rate $\eta_l \le \frac{1}{8LK}$ satisfies the following condition:*

$$\eta_l \le \frac{1}{6K}\min\left\{\left[\frac{\tau}{GL}\right]^{1/2}, \left[\frac{\tau^2}{GL^3\eta}\right]^{1/4}\right\}.$$

*Then for the iterates of Algorithm 4 (*FEDADAM*), we have the following:*

$$\min_{0\le t\le T-1} \mathbb{E}\|\nabla f(x_t)\|^2 = \mathcal{O}\left(\frac{\sqrt{\beta_2}\eta_l KG + \tau}{\eta_l KT}\left(\Psi + \Psi_{\text{var}}\right)\right).$$

where

$$\Psi = \frac{f(x_0)-f(x^*)}{\eta} + \frac{5\eta_l^3 K^2 L^2 T}{2\tau}\sigma^2,$$
$$\Psi_{var} = \left(G + \frac{\eta L}{2}\right)\left[\frac{4\eta_l^2 KT}{m\tau^2}\sigma_l^2 + \frac{20\eta_l^4 K^3 L^2 T}{\tau^2}\sigma^2\right].$$

Similar to the FEDADAGRAD case, we restate the above result for a specific choice of η_l, η and τ in order to highlight the dependence of K and T.

Corollary 3. *Suppose η_l is chosen such that conditions in Theorem 2 are satisfied. Furthermore, suppose client learning rate $\eta_l = \Theta(1/(KL\sqrt{T}))$, server learning rate $\eta = \Theta(\sqrt{Km})$ and $\tau = G/L$, then for sufficiently large T, we have for Algorithm 4 (*FEDADAM*) that $\min_{0\le t\le T-1}\mathbb{E}\|\nabla f(x_t)\|^2$ is*

$$\mathcal{O}\left(\frac{f(x_0)-f(x^*)}{\sqrt{mKT}} + \frac{2\sigma^2 L}{G^2\sqrt{mKT}} + \frac{\sigma^2}{GKT} + \frac{\sigma^2 L\sqrt{m}}{G^2\sqrt{K}T^{3/2}}\right).$$

The above convergence rate is similar to that of FEDADAGRAD. Thus, the discussion of convergence rates for FEDADAGRAD is also relevant here. We note that the convergence rate of generalized FEDAVG can also be obtained from the above analysis. In particular, when η, τ are large and η_l is small, the algorithm roughly corresponds to the generalized version of FEDAVG. Hence, our analysis also provides convergence rates of FEDAVG as a consequence. A more general result to capture other gradient-based server optimizers is left as a future work.

We end this section with a brief discussion about two aspects: (1) communication efficiency of the algorithms and (2) limited participation of the clients. First, note that the communication complexity of the algorithms is $\Theta(T)$. By Corollary 1 and 3, it is clear that a larger K leads to fewer rounds of communication when heterogeneity $\sigma_g = 0$. However, this is not necessarily the case when the clients are heterogeneous. Second, as mentioned earlier, for the sake of simplicity, our convergence analysis assumes full-participation i.e., $\mathcal{S} = [m]$. However, our analysis can be easily generalized to the case of limited participation at the cost of an additional variance term in the convergence rates that depends on the cardinality of the subset $\mathcal{S}$.

4 Experimental Evaluation: Datasets, Tasks, and Methods

We present an empirical evaluation of the federated optimization algorithms introduced in Sections 2 and 3 on what we believe is the most extensive and representative suite of federated datasets and modeling tasks that has been presented to date. Complete code for the datasets, models, and optimization algorithms will be open-sourced, and we hope this work will help establish benchmarks for further progress on federated optimization.

Our goal, in addition to validating our theoretical results, is to understand how the use of adaptive server optimizers, momentum, and learning rate decay can help improve convergence, particularly with respect to the defacto standard approach of FEDAVG. To accomplish this, we conduct extensive simulations of federated learning on six diverse and representative learning tasks across four datasets. Notably, three of the four datasets use a naturally-arising client partitioning, highly representative of real-world federated learning problems. A detailed description of the entire experimental setup can be found in the Appendix.

Datasets, models, and tasks We provide brief descriptions of the datasets and models used in our experiments here (refer to Appendix B for complete details).

CIFAR-100 We introduce a federated version of CIFAR-100 (Krizhevsky and Hinton, 2009) by randomly partitioning the training data among 500 clients. While Hsu et al. (2019) used latent Dirichlet allocation (LDA) over the fine labels of CIFAR-100 to create a federated dataset, we use a hierarchical LDA process (known as Pachinko Allocation (Li and McCallum, 2006)) over the coarse and fine labels. This creates more realistic heterogeneity among clients. We train a modified ResNet-18 on this dataset, where the batch normalization layers are replaced by group normalization layers (Wu and He, 2018; Hsieh et al., 2019).

EMNIST (Cohen et al., 2017) consists of images of digits and English characters, with 62 total classes. Federated EMNIST (Caldas et al., 2018) partitions the digits by their author, introducing natural heterogeneity due to variance in writing styles. We perform two distinct tasks on EMNIST, autoencoder training (AE) and character recognition (CR) using a CNN. For EMNIST AE, we train the "MNIST" autoencoder (Zaheer et al., 2018).

Shakespeare (McMahan et al., 2017) is a language modeling dataset built from the collective works of William Shakespeare. In this dataset, each client corresponds to a speaking role with at least two lines. We train an RNN that embeds sequences of ASCII characters into a low dimensional space, and then passes them through two LSTM layers and a densely connected softmax output layer to predict the next ASCII character.

StackOverflow (TF-Federated, 2019) is a dataset consisting of question and answers from StackOverflow, along with metadata, including tags. The dataset contains 342,477 unique users which we use as clients. We perform two tasks on this dataset: tag prediction via logistic regression (StackOverflow LR) on a bag-of-words featurization, and next-word prediction (StackOverflow NWP). For StackOverflow NWP, we use an RNN that embeds words into a low dimensional space, feeding them through an LSTM layer and a densely connected softmax output layer.

Table 1: Training dataset statistics.

DATASET	CLIENTS	EXAMPLES
CIFAR-100	500	50,000
EMNIST-62	3,400	671,585
SHAKESPEARE	715	16,068
STACKOVERFLOW	342,477	135,818,730

Implementation We implement all algorithms in TensorFlow Federated (Ingerman and Ostrowski, 2019). The client sampling is done uniformly at random from all training clients, without replacement within a given round, but with replacement across rounds. There are two important practical points about our implementation. First, rather than using a constant K local steps per client, we do *E client epochs* of training over each client's local dataset at each round. Second, in order to account for the varying number of gradient steps per client, we take a weighted average of the client outputs Δ_i^t according to the number of training samples of the client. While this is not used in our theoretical analysis, this follows the approach of (McMahan et al., 2017), which has been shown to generally outperform uniform weighting. We provide an open-source implementation of these algorithms, as well as code to reproduce all experiments[2].

Optimizers and hyperparameter tuning In our experiments, we compare FEDADAGRAD, FEDADAM, and FEDYOGI to FEDAVG where the server learning rate η is tuned (we do not require $\eta = 1$) as well as when the server optimizer is SGD with a tuned learning rate and momentum parameter of 0.9. In a slight abuse of notation, we refer to the former as FEDAVG and the latter as FEDAVGM. We select η_l and η by tuning over moderately sized grids (see Appendix C.2 for the full grids). We fix the batch size on a per-task level (see Appendix C.3 for a full accounting). For FEDYOGI, we set the initial accumulator value according to the strategy outlined for YOGI by Zaheer et al. (2018). For a full description of the initialization, see Appendix C.5. We do not tune any other hyperparameters throughout.

In real cross-device FL deployments, communication bandwidth is a fundamental constraint (Bonawitz et al., 2019; Kairouz et al., 2019). Since all algorithms exchange equal-sized objects between server and clients, we use the number of rounds of communication as an appropriate proxy for wall-clock training time in realistic FL deployments.

5 Experimental Evaluation: Results

In this section, we compare the performance of FEDADAGRAD, FEDADAM, FEDYOGI, FEDAVG, and FEDAVGM on various tasks. We perform our experiments in two general regimes: when client and server learning rates (η_l and η in Algorithm 2) are constant, and when η_l varies with respect to the round number.

5.1 Constant learning rates

We investigate whether adaptive optimization alone (that is, with tuned but constant η_l and η) is enough to significantly improve the performance of FEDAVG. For each optimizer, we select η_l and η based on the average validation performance over the last 100 communication rounds. Appendix C gives the learning rates grids and selected learning rates for all experiments. Table 2 summarizes the last-100-round performance for all experiments. We discuss our empirical results, especially differences in performance between tasks and optimizers, in full detail below.

Sparse-gradient tasks Text data naturally produces long-tailed feature distributions, often leading to sparse or approximately-sparse gradients which adaptive optimizers can capitalize on. This has been observed many times for centralized training, and we show the intuition also applies to FL. Both the LR and NWP tasks on StackOverflow

[2] https://github.com/tensorflow/federated/tree/master/tensorflow_federated/python/research/optimization

Table 2: Average validation performance over the last 100 rounds: % accuracy for rows 1–4; Recall@5 for StackOverflow LR; and MSE for EMNIST AE. Performance within 0.5%/0.05/0.25 of the best result are shown in bold.

FED...	ADAGRAD	ADAM	YOGI	AVGM	AVG
CIFAR-100	23.9	**42.3**	41.6	37.3	26.5
EMNIST CR	**85.7**	**86.0**	**86.1**	**86.3**	85.9
SHAKESPEARE	57.1	**57.4**	**57.6**	**57.5**	57.0
STACKOV... NWP	11.3	**22.1**	**22.2**	13.7	9.5
STACKOV... LR	**0.68**	0.62	0.64	0.22	0.19
EMNIST AE	7.29	16.99	**0.98**	**1.21**	2.63

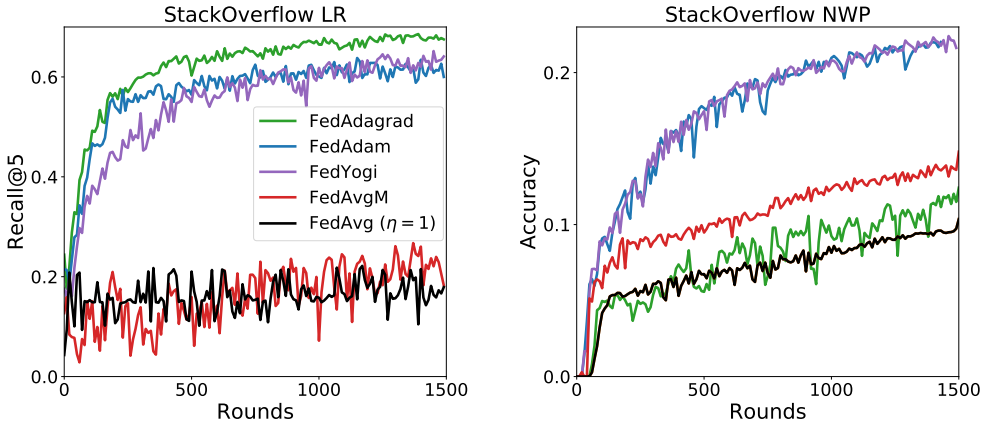

Figure 1: Test set performance using constant learning rates η and η_l tuned to achieve the best performance on the last 100 communication rounds; see Appendix C.2 for grids. LR sampled 10 clients per round and 10 local epoch and NWP sampled 50 clients per round with 1 local epoch.

exhibit such behavior, though we emphasize that these two tasks are otherwise dramatically different—in feature representation (bag-of-words vs variable-length token sequence), model architecture (GLM vs deep network), and optimization landscape (convex vs non-convex).

The StackOverflow tag-prediction LR task uses a bag-of-words representation on the 10^5 most common words in the corpus, and StackoverFlow NWP uses a 96-dimensional embedding vector for each word. In both cases, words that do not appear in a particular client's set of examples will produce a zero (for LR) or near-zero (for NWP, due to the softmax) client updates. Thus, the accumulator $v_{t,j}$ in Algorithm 3 and 4 will remain small for parameters tied to rare words, allowing large updates to be made when they do occur; conversely, $v_{t,j}$ will grow quickly for common words, preventing the parameter pertaining to those words from oscillating. This intuition is born out by Figure 1, where the adaptive optimizers dramatically outperform their non-adaptive competitors. For the non-convex NWP task, the inclusion of momentum also appears to be critical, whereas it slightly hinders performance for the convex LR task.

Momentum doesn't help in convex LR case...

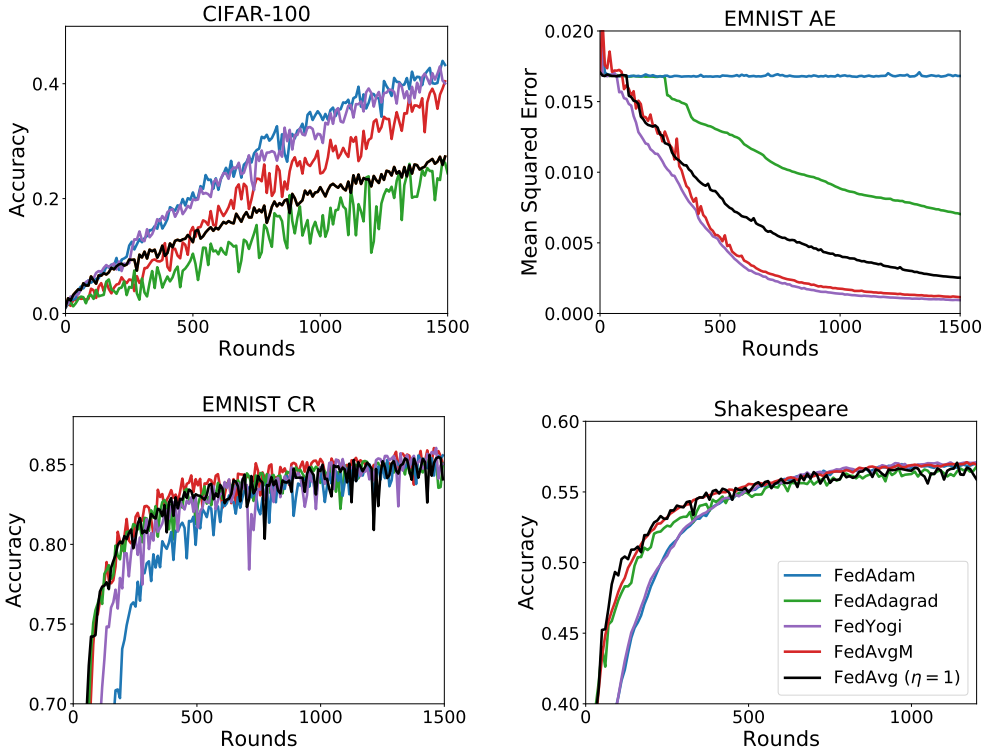

Figure 2: Test performance under the same setting as Figure 1, but with one local client epoch, and 10 clients sampled per round.

Dense-gradient tasks The results for CIFAR-100, EMNIST AE/CR and Shakespeare are given in Figure 2. These tasks lack the sparse-gradient structure of the two previous tasks (see Appendix B for architecture and featurization details). For EMNIST AE, we normalize the MSE by the output dimension 28×28 throughout. When using FEDAVG, we found that a server learning rate of 1 worked best for all tasks (corresponding to the original FEDAVG proposed by McMahan et al. (2017)).

Two of the tasks, EMNIST CR and Shakespeare, are relatively easy—all optimizers perform well *once suitably tuned*. For the two more challenging tasks (training ResNet-18 on CIFAR-100 and training a bottleneck autoencoder on EMNIST), we see that adaptive server optimizers and momentum can offer a substantial improvement over vanilla FEDAVG. In particular, FEDYOGI was the only optimizer to perform well in all settings. FEDAVGM was outperformed by FEDADAM and FEDYOGI on CIFAR-100, and FEDADAM failed to converge for any server/client learning rate combinations on EMNIST AE. We note that the bottleneck architecture used tends to create saddle points that can be difficult to escape.

On the ratio of η to η_l As suggested by Theorem 1, we expect an inverse relationship between the client learning rate and the best corresponding server learning rate. Intuitively, if the clients learning rate is large, the server must account for drift by reducing its learning rate. To validate this, we plot the optimal server learning rate for each client learning rate in the tuning grids for each task, with results in Figure 3. Our results show that for most tasks, there is a clear inverse relationship. There are a few exceptions. In StackOverflow LR, there seemed to be a "preferred" server learning rate for each optimizer, while in EMNIST AE the learning rates exhibited a more complicated relationship. Note that Corollary 1 suggests an inverse relationship only for approaching a critical point, not for escaping saddle points (the primary obstacle in EMNIST AE).

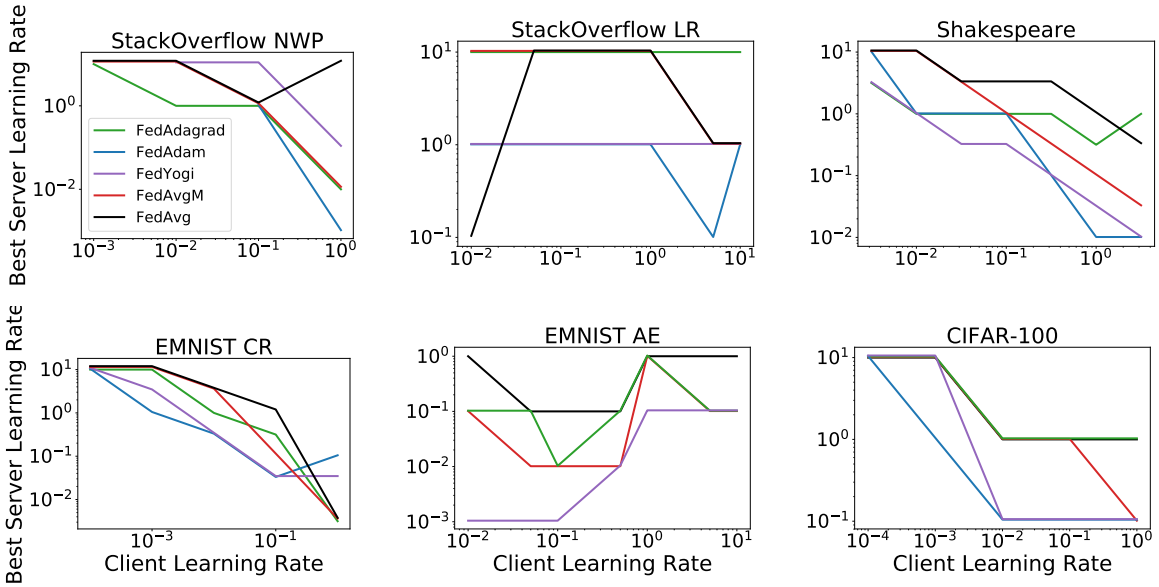

Figure 3: The best server learning rate in our hyperparameter tuning grids for each client learning rate, optimizers, and task. The experimental setup for each task is identical to that in Figures 1 and 2. We select the server learning rates based on the average performance over the last 100 communication rounds.

5.2 Improving performance with learning rate decay

Despite the success in the previous section, it is natural to ask if there is still more to be gained. To test this, we trained the EMNIST CR model in a centralized fashion on a shuffled version of the dataset (taking a union over all client datasets). We trained for 100 epochs and again carefully tuned learning rates for each (centralized) optimizer, achieving an accuracy of 88% (see Table 3, CENTRALIZED row), significantly above the best results from Table 2.

The theoretical results in Section 3 point to a partial explanation, as they only hold when the client learning rate is small (on the order of $1/\sqrt{T}$) or is decayed at a rate of $1/\sqrt{t}$, in order to combat client drift. To test this, we perform federated experiments using two client learning rate schedules, INVSQRT (an inverse square root decay of $\eta_l/\sqrt{t}$) and EXPDECAY (a "staircase" exponential decay schedule where η_l is decreased by a factor of 0.1 after every 500 rounds (Krizhevsky, 2014; Goyal et al., 2017)). We use 10 client epochs, and sample 10 clients per round. Table 3 (lower rows) gives the result. We find that EXPDECAY significantly improves the accuracy of all federated optimizers.

Client LR schedules.

Table 3: (Top) Test accuracy (%) of a model trained centrally with various optimizers. (Bottom) Average validation accuracy (%) over the last 100 rounds of various federated optimizers on the EMNIST character recognition task, for three schedules for η_l. Accuracies within 0.5% of the best result as shown in bold.

	ADAGRAD	ADAM	YOGI	SGDM	SGD
CENTRALIZED	88.0	87.9	88.0	87.7	87.7
FED...	**ADAGRAD**	**ADAM**	**YOGI**	**AVGM**	**AVG**
CONSTANT η_l	85.7	86.0	86.1	86.3	85.9
INVSQRT	84.8	86.8	86.6	86.1	85.5
EXPDECAY	87.1	87.1	**87.6**	**87.5**	87.1

and allows them to get close to the best centralized accuracy. FEDYOGI and FEDAVGM with EXPDECAY outperform all other federated methods, especially methods employing constant client learning rates.

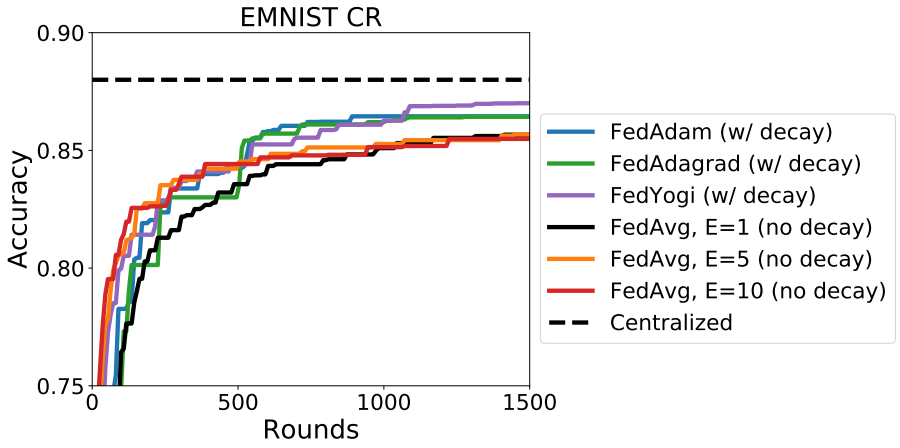

Figure 4: The maximum test accuracy over time of various optimizers on EMNIST CR, as well as the best centralized accuracy.

We then compare the performance of adaptive optimizers (using EXPDECAY) with vanilla FEDAVG (without EXPDECAY). For FEDAVG, we vary the number of local client epochs E over $\{1, 5, 10\}$, while we use 10 local client epochs for the adaptive optimizers. In all cases, we use 10 clients per round. We also compare to the best centralized accuracy in Table 3. The results are in Figure 4.

While FEDAVG with $E = 10$ achieved good performance initially, it obtained slightly worse accuracy than $E \in \{1, 5\}$ after enough rounds. On the other hand, the adaptive optimizers with EXPDECAY surpass all three versions of FEDAVG after the first client learning rate decay occurs, and only perform better thereafter.

decay helps in performance.

6 Conclusion

In this paper, we demonstrated that adaptive optimizers can be powerful tools in improving the convergence of federated learning. Moreover, we showed that by using a simple client/server optimizer framework, we can apply adaptive optimizers to federated learning in a principled and intuitive manner. While our work establishes the theoretical and empirical benefits of this framework, it also raises many interesting questions about how best to perform federated optimization. Example directions for future research include exploring the use of adaptive client optimizers and developing comprehensive heuristics for setting learning rates and other hyperparameters.

(1) (2) Promising future directions !

References

Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloe Kiddon, Jakub Konečný, Stefano Mazzocchi, H Brendan McMahan, et al. Towards federated learning at scale: System design. *arXiv preprint arXiv:1902.01046*, 2019.

Sebastian Caldas, Peter Wu, Tian Li, Jakub Konečný, H Brendan McMahan, Virginia Smith, and Ameet Talwalkar. LEAF: A benchmark for federated settings. *arXiv preprint arXiv:1812.01097*, 2018.

Gregory Cohen, Saeed Afshar, Jonathan Tapson, and Andre Van Schaik. EMNIST: Extending MNIST to handwritten letters. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 2921–2926. IEEE, 2017.

John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

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

Kevin Hsieh, Amar Phanishayee, Onur Mutlu, and Phillip B Gibbons. The non-IID data quagmire of decentralized machine learning. *arXiv preprint arXiv:1910.00189*, 2019.

Tzu-Ming Harry Hsu, Hang Qi, and Matthew Brown. Measuring the effects of non-identical data distribution for federated visual classification. *arXiv preprint arXiv:1909.06335*, 2019.

Alex Ingerman and Krzys Ostrowski. Introducing TensorFlow Federated, 2019. URL https://medium.com/tensorflow/introducing-tensorflow-federated-a4147aa20041.

Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Keith Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. Advances and open problems in federated learning. *arXiv preprint arXiv:1912.04977*, 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. *arXiv preprint arXiv:1910.06378*, 2019.

Ahmed Khaled, Konstantin Mishchenko, and Peter Richtárik. First analysis of local GD on heterogeneous data. *arXiv preprint arXiv:1909.04715*, 2019.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

Alex Krizhevsky. One weird trick for parallelizing convolutional neural networks. *arXiv preprint arXiv:1404.5997*, 2014.

Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

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

Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. Federated learning: Challenges, methods, and future directions. *arXiv preprint arXiv:1908.07873*, 2019a.

Wei Li and Andrew McCallum. Pachinko allocation: DAG-structured mixture models of topic correlations. In *Proceedings of the 23rd international conference on Machine learning*, pages 577–584, 2006.

Xiang Li, Kaixuan Huang, Wenhao Yang, Shusen Wang, and Zhihua Zhang. On the convergence of FedAvg on non-IID data. *arXiv preprint arXiv:1907.02189*, 2019b.

Xiaoyu Li and Francesco Orabona. On the convergence of stochastic gradient descent with adaptive stepsizes. *arXiv preprint arXiv:1805.08114*, 2018.

Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, AISTATS 2017, 20-22 April 2017, Fort Lauderdale, FL, USA*, pages 1273–1282, 2017. URL http://proceedings.mlr.press/v54/mcmahan17a.html.

H. Brendan McMahan and Matthew J. Streeter. Adaptive bound optimization for online convex optimization. In *COLT The 23rd Conference on Learning Theory*, 2010.

Sashank J Reddi, Ahmed Hefny, Suvrit Sra, Barnabás Póczós, and Alex Smola. Stochastic variance reduction for nonconvex optimization. *arXiv:1603.06160*, 2016.

Sashank J Reddi, Satyen Kale, and Sanjiv Kumar. On the convergence of Adam and beyond. *arXiv preprint arXiv:1904.09237*, 2019.

Sebastian U. Stich. Local SGD converges fast and communicates little. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=S1g2JnRcFX.

Sebastian U Stich and Sai Praneeth Karimireddy. The error-feedback framework: Better rates for SGD with delayed gradients and compressed communication. *arXiv preprint arXiv:1909.05350*, 2019.

TF-Federated. TensorFlow Federated Stack Overflow dataset, 2019. URL https://www.tensorflow.org/federated/api_docs/python/tff/simulation/datasets/stackoverflow/load_data.

Jianyu Wang and Gauri Joshi. Cooperative SGD: A unified framework for the design and analysis of communication-efficient SGD algorithms. *arXiv preprint arXiv:1808.07576*, 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.

Rachel Ward, Xiaoxia Wu, and Leon Bottou. Adagrad stepsizes: Sharp convergence over nonconvex landscapes, from any initialization. *arXiv preprint arXiv:1806.01811*, 2018.

Xiaoxia Wu, Simon S Du, and Rachel Ward. Global convergence of adaptive gradient methods for an over-parameterized neural network. *arXiv preprint arXiv:1902.07111*, 2019.

Yuxin Wu and Kaiming He. Group normalization. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 3–19, 2018.

Hao Yu, Sen Yang, and Shenghuo Zhu. Parallel restarted SGD with faster convergence and less communication: Demystifying why model averaging works for deep learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 5693–5700, 2019.

Manzil Zaheer, Sashank Reddi, Devendra Sachan, Satyen Kale, and Sanjiv Kumar. Adaptive methods for nonconvex optimization. In *Advances in Neural Information Processing Systems*, pages 9815–9825, 2018.

Jingzhao Zhang, Sai Praneeth Karimireddy, Andreas Veit, Seungyeon Kim, Sashank J. Reddi, Sanjiv Kumar, and Suvrit Sra. Why ADAM beats SGD for attention models. *arXiv preprint arxiv:1912.03194*, 2019.

Martin Zinkevich, Markus Weimer, Lihong Li, and Alex J Smola. Parallelized stochastic gradient descent. In *Advances in neural information processing systems*, pages 2595–2603, 2010.

A Proof of results

A.1 Theorem 1

Proof of Theorem 1. Recall that the server update of FEDADAGRAD is the following

$$x_{t+1,i} = x_{t,i} + \eta \frac{\Delta_{t,i}}{\sqrt{v_{t,i}} + \tau},$$

for all $i \in [d]$. Since the function f is L-smooth, we have the following:

$$\begin{aligned} f(x_{t+1}) &\leq f(x_t) + \langle \nabla f(x_t), x_{t+1} - x_t \rangle + \frac{L}{2}\|x_{t+1} - x_t\|^2 \\ &= f(x_t) + \eta \left\langle \nabla f(x_t), \frac{\Delta_t}{\sqrt{v_t} + \tau} \right\rangle + \frac{\eta^2 L}{2} \sum_{i=1}^{d} \frac{\Delta_{t,i}^2}{(\sqrt{v_{t,i}} + \tau)^2} \end{aligned} \tag{2}$$

The second step follows simply from FEDADAGRAD's update. We take the expectation of $f(x_{t+1})$ (over randomness at time step t) in the above inequality:

$$\begin{aligned} \mathbb{E}_t[f(x_{t+1})] &\leq f(x_t) + \eta \left\langle \nabla f(x_t), \mathbb{E}_t \left[\frac{\Delta_t}{\sqrt{v_t} + \tau} \right] \right\rangle + \frac{\eta^2 L}{2} \sum_{i=1}^{d} \mathbb{E}_t \left[\frac{\Delta_{t,i}^2}{(\sqrt{v_{t,i}} + \tau)^2} \right] \\ &= f(x_t) + \eta \left\langle \nabla f(x_t), \mathbb{E}_t \left[\frac{\Delta_t}{\sqrt{v_t} + \tau} - \frac{\Delta_t}{\sqrt{v_{t-1}} + \tau} + \frac{\Delta_t}{\sqrt{v_{t-1}} + \tau} \right] \right\rangle \\ &\qquad + \frac{\eta^2 L}{2} \sum_{j=1}^{d} \mathbb{E}_t \left[\frac{\Delta_{t,j}^2}{(\sqrt{v_{t,j}} + \tau)^2} \right] \\ &= f(x_t) + \eta \underbrace{\left\langle \nabla f(x_t), \mathbb{E}_t \left[\frac{\Delta_t}{\sqrt{v_{t-1}} + \tau} \right] \right\rangle}_{T_1} + \eta \underbrace{\left\langle \nabla f(x_t), \mathbb{E}_t \left[\frac{\Delta_t}{\sqrt{v_t} + \tau} - \frac{\Delta_t}{\sqrt{v_{t-1}} + \tau} \right] \right\rangle}_{T_2} \\ &\qquad + \frac{\eta^2 L}{2} \sum_{j=1}^{d} \mathbb{E}_t \left[\frac{\Delta_{t,j}^2}{(\sqrt{v_{t,j}} + \tau)^2} \right] \end{aligned} \tag{3}$$

We will first bound T_2 in the following manner:

$$\begin{aligned} T_2 &= \left\langle \nabla f(x_t), \mathbb{E}_t \left[\frac{\Delta_t}{\sqrt{v_t} + \tau} - \frac{\Delta_t}{\sqrt{v_{t-1}} + \tau} \right] \right\rangle \\ &= \mathbb{E}_t \sum_{j=1}^{d} [\nabla f(x_t)]_j \times \left[\frac{\Delta_{t,j}}{\sqrt{v_{t,j}} + \tau} - \frac{\Delta_{t,j}}{\sqrt{v_{t-1,j}} + \tau} \right] \\ &= \mathbb{E}_t \sum_{j=1}^{d} [\nabla f(x_t)]_j \times \Delta_{t,j} \times \left[\frac{\sqrt{v_{t-1,j}} - \sqrt{v_{t,j}}}{(\sqrt{v_{t,j}} + \tau)(\sqrt{v_{t-1,j}} + \tau)}, \right] \end{aligned}$$

and recalling $v_t = v_{t-1} + \Delta_t^2$ so $-\Delta_{t,j}^2 = (\sqrt{v_{t-1,j}} - \sqrt{v_{t,j}})(\sqrt{v_{t-1,j}} + \sqrt{v_{t,j}})$ we have,

$$\begin{aligned} &= \mathbb{E}_t \sum_{j=1}^{d} [\nabla f(x_t)]_j \times \Delta_{t,j} \times \left[\frac{-\Delta_{t,j}^2}{(\sqrt{v_{t,j}} + \tau)(\sqrt{v_{t-1,j}} + \tau)(\sqrt{v_{t-1,j}} + \sqrt{v_{t,j}})} \right] \\ &\leq \mathbb{E}_t \sum_{j=1}^{d} |[\nabla f(x_t)]_j| \times |\Delta_{t,j}| \times \left[\frac{\Delta_{t,j}^2}{(\sqrt{v_{t,j}} + \tau)(\sqrt{v_{t-1,j}} + \tau)(\sqrt{v_{t-1,j}} + \sqrt{v_{t,j}})} \right] \\ &\leq \mathbb{E}_t \sum_{j=1}^{d} |[\nabla f(x_t)]_j| \times |\Delta_{t,j}| \times \left[\frac{\Delta_{t,j}^2}{(v_{t,j} + \tau^2)(\sqrt{v_{t-1,j}} + \tau)} \right] \qquad \text{since } v_{t-1,j} \geq \tau^2. \end{aligned}$$

Here $v_{t-1,j} \geq \tau$ since $v_{-1} \geq \tau$ (see the initialization of Algorithm 3) and $v_{t,j}$ is increasing in t. The above bound can be further upper bounded in the following manner:

$$\begin{aligned} T_2 &\leq \mathbb{E}_t \sum_{j=1}^{d} \eta_l K G^2 \left[\frac{\Delta_{t,j}^2}{(v_{t,j}+\tau^2)(\sqrt{v_{t-1,j}}+\tau)} \right] && \text{since } [\nabla f(x_t)]_i \leq G \text{ and } \Delta_{t,i} \leq \eta_l K G \\ &\leq \mathbb{E}_t \sum_{j=1}^{d} \frac{\eta_l K G^2}{\tau} \left[\frac{\Delta_{t,j}^2}{\sum_{l=0}^{t} \Delta_{l,j}^2 + \tau^2} \right] && \text{since } \sqrt{v_{t-1,j}} \geq 0. \end{aligned} \tag{4}$$

Bounding T_1 We now turn our attention to bounding the term T_1, which we need to be sufficiently negative. We observe the following:

$$\begin{aligned} T_1 &= \left\langle \nabla f(x_t), \mathbb{E}_t \left[\frac{\Delta_t}{\sqrt{v_{t-1}}+\tau} \right] \right\rangle \\ &= \left\langle \frac{\nabla f(x_t)}{\sqrt{v_{t-1}}+\tau}, \mathbb{E}_t \left[\Delta_t - \eta_l K \nabla f(x_t) + \eta_l K \nabla f(x_t) \right] \right\rangle \\ &= -\eta_l K \sum_{j=1}^{d} \frac{[\nabla f(x_t)]_j^2}{\sqrt{v_{t-1,j}}+\tau} + \underbrace{\left\langle \frac{\nabla f(x_t)}{\sqrt{v_{t-1}}+\tau}, \mathbb{E}_t \left[\Delta_t + \eta_l K \nabla f(x_t) \right] \right\rangle}_{T_3}. \end{aligned} \tag{5}$$

In order to bound T_1 , we use the following upper bound on T_3 (which captures the difference between the actual update Δ_t and an appropriate scaling of $-\nabla f(x_t)$):

$$\begin{aligned} T_3 &= \left\langle \frac{\nabla f(x_t)}{\sqrt{v_{t-1}}+\tau}, \mathbb{E}_t \left[\Delta_t + \eta_l K \nabla f(x_t) \right] \right\rangle \\ &= \left\langle \frac{\nabla f(x_t)}{\sqrt{v_{t-1}}+\tau}, \mathbb{E}_t \left[-\frac{1}{m} \sum_{i=1}^{m} \sum_{k=0}^{K-1} \eta_l g_{i,k}^t + \eta_l K \nabla f(x_t) \right] \right\rangle \\ &= \left\langle \frac{\nabla f(x_t)}{\sqrt{v_{t-1}}+\tau}, \mathbb{E}_t \left[-\frac{1}{m} \sum_{i=1}^{m} \sum_{k=0}^{K-1} \eta_l \nabla F_i(x_{i,k}^t) + \eta_l K \nabla f(x_t) \right] \right\rangle. \end{aligned}$$

Here we used the fact that $\nabla f(x_t) = \frac{1}{m} \sum_{i=1}^{m} \nabla F_i(x_t)$ and $g_{i,k}^t$ is an unbiased estimator of the gradient at $x_{i,k}^t$, we further bound T_3 as follows using a simple application of the fact that $ab \leq (a^2+b^2)/2$. :

$$\begin{aligned} T_3 &\leq \frac{\eta_l K}{2} \sum_{j=1}^{d} \frac{[\nabla f(x_t)]_j^2}{\sqrt{v_{t-1,j}}+\tau} + \frac{\eta_l}{2K} \mathbb{E}_t \left[\left\| \frac{1}{m} \sum_{i=1}^{m} \sum_{k=0}^{K-1} \frac{\nabla F_i(x_{i,k}^t)}{\sqrt{\sqrt{v_{t-1}}+\tau}} - \frac{1}{m} \sum_{i=1}^{m} \sum_{k=0}^{K-1} \frac{\nabla F_i(x_t)}{\sqrt{\sqrt{v_{t-1}}+\tau}} \right\|^2 \right] \\ &\leq \frac{\eta_l K}{2} \sum_{j=1}^{d} \frac{[\nabla f(x_t)]_j^2}{\sqrt{v_{t-1,j}}+\tau} + \frac{\eta_l}{2m} \mathbb{E}_t \left[\sum_{i=1}^{m} \sum_{k=0}^{K-1} \left\| \frac{\nabla F_i(x_{i,k}^t) - \nabla F_i(x_t)}{\sqrt{\sqrt{v_{t-1}}+\tau}} \right\|^2 \right] \\ &\leq \frac{\eta_l K}{2} \sum_{j=1}^{d} \frac{[\nabla f(x_t)]_j^2}{\sqrt{v_{t-1,j}}+\tau} + \frac{\eta_l L^2}{2m\tau} \mathbb{E}_t \left[\sum_{i=1}^{m} \sum_{k=0}^{K-1} \|x_{i,k}^t - x_t\|^2 \right] \qquad \text{using Assumption 1 and } v_{t-1} \geq 0. \end{aligned} \tag{6}$$

The second inequality follows from Lemma 7. The last inequality follows from L-Lipschitz nature of the gradient (Assumption 1). We now prove a lemma that bounds the "drift" of the $x_{i,k}^t$ from x_t:

Lemma 4. *For any step-size satisfying $\eta_l \leq \frac{1}{8LK}$, we can bound the drift for any $k \in \{0, \cdots, K-1\}$ as*

$$\frac{1}{m} \sum_{i=1}^{m} \mathbb{E} \|x_{i,k}^t - x_t\|^2 \leq 5K\eta_l^2 \mathbb{E} \sum_{j=1}^{d} (\sigma_{l,j}^2 + 2K\sigma_{g,j}^2) + 10K^2\eta_l^2 \mathbb{E}[\|\nabla f(x_t))\|^2]. \tag{7}$$

Proof. The result trivially holds for $k = 1$ since $x_{i,0}^t = x_t$ for all $i \in [m]$. We now turn our attention to the case where $k \geq 1$. To prove the above result, we observe that for any client $i \in [m]$ and $k \in [K]$,

$$\begin{aligned}
&\mathbb{E}\|x_{i,k}^t - x_t\|^2 = \mathbb{E}\left\|x_{i,k-1}^t - x_t - \eta_l g_{i,k-1}^t\right\|^2 \\
&\leq \mathbb{E}\left\|x_{i,k-1}^t - x_t - \eta_l(g_{i,k-1}^t - \nabla F_i(x_{i,k-1}^t) + \nabla F_i(x_{i,k-1}^t) - \nabla F_i(x_t) + \nabla F_i(x_t) - \nabla f(x_t) + \nabla f(x_t))\right\|^2 \\
&\leq \left(1 + \frac{1}{2K-1}\right)\mathbb{E}\left\|x_{i,k-1}^t - x_t\right\|^2 + \mathbb{E}\left\|\eta_l(g_{i,k-1}^t - \nabla F_i(x_{i,k-1}^t))\right\|^2 \\
&\qquad + 6K\mathbb{E}[\|\eta_l(\nabla F_i(x_{i,k-1}^t) - \nabla F_i(x_t))\|^2] + 6K\mathbb{E}[\|\eta_l(\nabla F_i(x_t) - \nabla f(x_t))\|^2] + 6K\mathbb{E}[\|\eta_l \nabla f(x_t)))\|^2]
\end{aligned}$$

The first inequality uses the fact that $g_{k-1,i}^t$ is an unbiased estimator of $\nabla F_i(x_{i,k-1}^t)$ and Lemma 8. The above quantity can be further bounded by the following:

$$\begin{aligned}
\mathbb{E}\|x_{i,k}^t - x_t\|^2 &\leq \left(1 + \frac{1}{2K-1}\right)\mathbb{E}\|x_{i,k-1}^t - x_t\|^2 + \eta_l^2\mathbb{E}\sum_{j=1}^d \sigma_{l,j}^2 + 6K\eta_l^2\mathbb{E}\|L(x_{i,k-1}^t - x_t)\|^2 \\
&\qquad + 6K\mathbb{E}[\|\eta_l(\nabla F_i(x_t) - \nabla f(x_t))\|^2] + 6K\mathbb{E}[\|\eta_l \nabla f(x_t)))\|^2] \\
&= \left(1 + \frac{1}{2K-1} + 6K\eta_l^2 L^2\right)\mathbb{E}\|(x_{i,k-1}^t - x_t)\|^2 + \eta_l^2\mathbb{E}\sum_{j=1}^d \sigma_{l,j}^2 \\
&\qquad + 6K\mathbb{E}[\|\eta_l(\nabla F_i(x_t) - \nabla f(x_t))\|^2] + 6K\eta_l^2\mathbb{E}[\|\nabla f(x_t)))\|^2]
\end{aligned}$$

Here, the first inequality follows from Assumption 1 and 2. Averaging over the clients i, we obtain the following:

$$\begin{aligned}
\frac{1}{m}\sum_{i=1}^m \mathbb{E}\|x_{i,k}^t - x_t\|^2 &\leq \left(1 + \frac{1}{2K-1} + 6K\eta_l^2 L^2\right)\frac{1}{m}\sum_{i=1}^m \mathbb{E}\|x_{i,k-1}^t - x_t\|^2 + \eta_l^2\mathbb{E}\sum_{j=1}^d \sigma_{l,j}^2 \\
&\qquad + \frac{6K}{m}\sum_{i=1}^m \mathbb{E}[\|\eta_l(\nabla F_i(x_t) - \nabla f(x_t))\|^2] + 6K\eta_l^2\mathbb{E}[\|\nabla f(x_t)))\|^2] \\
&\leq \left(1 + \frac{1}{2K-1} + 6K\eta_l^2 L^2\right)\frac{1}{m}\sum_{i=1}^m \mathbb{E}\|x_{i,k-1}^t - x_t\|^2 + \eta_l^2\mathbb{E}\sum_{j=1}^d (\sigma_{l,j}^2 + 6K\sigma_{g,j}^2) \\
&\qquad + 6K\eta_l^2\mathbb{E}[\|\nabla f(x_t)))\|^2]
\end{aligned}$$

From the above, we get the following inequality:

$$\begin{aligned}
\frac{1}{m}\sum_{i=1}^m \mathbb{E}\|x_{i,k}^t - x_t\|^2 &\leq \left(1 + \frac{1}{K-1}\right)\frac{1}{m}\sum_{i=1}^m \mathbb{E}\|x_{i,k-1}^t - x_t\|^2 + \eta_l^2\mathbb{E}\sum_{j=1}^d (\sigma_{l,j}^2 + 2K\sigma_{g,j}^2) \\
&\qquad + 2K\eta_l^2\mathbb{E}[\|\nabla f(x_t)))\|^2]
\end{aligned}$$

Unrolling the recursion, we obtain the following:

$$\begin{aligned}
&\frac{1}{m}\sum_{i=1}^m \mathbb{E}\|x_{i,k}^t - x_t\|^2 \leq \sum_{p=0}^{k-1}\left(1 + \frac{1}{K-1}\right)^p \left[\eta_l^2\mathbb{E}\sum_{j=1}^d (\sigma_{l,j}^2 + 6K\sigma_{g,j}^2) + 6K\eta_l^2\mathbb{E}[\|\nabla f(x_t)))\|^2]\right] \\
&\leq (K-1) \times \left[\left(1 + \frac{1}{K-1}\right)^K - 1\right] \times \left[\eta_l^2\mathbb{E}\sum_{j=1}^d (\sigma_{l,j}^2 + 6K\sigma_{g,j}^2) + 6K\eta_l^2\mathbb{E}[\|\nabla f(x_t)))\|^2]\right] \\
&\leq \left[5K\eta_l^2\mathbb{E}\sum_{j=1}^d (\sigma_{l,j}^2 + 6K\sigma_{g,j}^2) + 30K^2\eta_l^2\mathbb{E}[\|\nabla f(x_t)))\|^2]\right],
\end{aligned}$$

concluding the proof of Lemma 4. The last inequality uses the fact that $(1 + \frac{1}{K-1})^K \leq 5$ for $K > 1$. □

Using the above lemma in Equation 6 and the fact that $\eta_l \leq \frac{1}{8KL}$, we can bound T_3 in the following manner:

$$\begin{aligned}
T_3 &\leq \frac{\eta_l K}{2}\sum_{j=1}^{d}\frac{[\nabla f(x_t)]_j^2}{\sqrt{v_{t-1,j}}+\tau} + \frac{\eta_l L^2}{2m\tau}\mathbb{E}_t\left[\sum_{i=1}^{m}\sum_{k=0}^{K-1}\sum_{j=1}^{d}([x_{i,k}^t]_j - [x_t]_j)^2\right] \\
&\leq \frac{\eta_l K}{2}\sum_{j=1}^{d}\frac{[\nabla f(x_t)]_j^2}{\sqrt{v_{t-1,j}}+\tau} + \frac{\eta_l K L^2}{2\tau}\left[5K\eta_l^2\mathbb{E}\sum_{j=1}^{d}(\sigma_{l,j}^2 + 6K\sigma_{g,j}^2) + \frac{5}{32L^2}\mathbb{E}[\|\nabla f(x_t)))\|^2]\right] \\
&\leq \frac{3\eta_l K}{4}\sum_{j=1}^{d}\frac{[\nabla f(x_t)]_j^2}{\sqrt{v_{t-1,j}}+\tau} + \frac{5\eta_l^3 K^2 L^2}{2\tau}\mathbb{E}\sum_{j=1}^{d}(\sigma_{l,j}^2 + 6K\sigma_{g,j}^2)
\end{aligned}$$

Using the above bound in Equation 5, we get

$$T_1 \leq -\frac{\eta_l K}{4}\sum_{j=1}^{d}\frac{[\nabla f(x_t)]_j^2}{\sqrt{v_{t-1,j}}+\tau} + \frac{5\eta_l^3 K^2 L^2}{2\tau}\mathbb{E}\sum_{j=1}^{d}(\sigma_{l,j}^2 + 6K\sigma_{g,j}^2) \tag{8}$$

Putting the pieces together Substituting in Equation (3), bounds T_1 in Equation (8) and bound T_2 in Equation (4), we obtain

$$\begin{aligned}
\mathbb{E}_t[f(x_{t+1})] \leq f(x_t) + \eta \times &\left[-\frac{\eta_l K}{4}\sum_{j=1}^{d}\frac{[\nabla f(x_t)]_j^2}{\sqrt{v_{t-1,j}}+\tau} + \frac{5\eta_l^3 K^2 L^2}{2\tau}\mathbb{E}\sum_{j=1}^{d}(\sigma_{l,j}^2 + 6K\sigma_{g,j}^2)\right] \\
&+ \eta \times \mathbb{E}\sum_{j=1}^{d}\frac{\eta_l K G^2}{\tau}\left[\frac{\Delta_{t,j}^2}{\sum_{l=0}^{t}\Delta_{l,j}^2 + \tau^2}\right] + \frac{\eta^2}{2}\sum_{j=1}^{d}L\mathbb{E}\left[\frac{\Delta_{t,j}^2}{\sum_{l=0}^{t}\Delta_{l,j}^2 + \tau^2}\right].
\end{aligned} \tag{9}$$

Rearranging the above inequality and summing it from $t = 0$ to $T-1$, we get

$$\begin{aligned}
\sum_{t=0}^{T-1}\frac{\eta_l K}{4}\sum_{j=1}^{d}\mathbb{E}\frac{[\nabla f(x_t)]_j^2}{\sqrt{v_{t-1,j}}+\tau} \leq \frac{f(x_0) - \mathbb{E}[f(x_T)]}{\eta} + \frac{5\eta_l^3 K^2 L^2 T}{2\tau}\sum_{j=1}^{d}(\sigma_{l,j}^2 + 6K\sigma_{g,j}^2) \\
+ \sum_{t=0}^{T-1}\mathbb{E}\sum_{j=1}^{d}\left(\frac{\eta_l K G^2}{\tau} + \frac{\eta L}{2}\right) \times \left[\frac{\Delta_{t,j}^2}{\sum_{l=0}^{t}\Delta_{l,j}^2 + \tau^2}\right]
\end{aligned} \tag{10}$$

The first inequality uses simple telescoping sum. For completing the proof, we need the following result.

Lemma 5. *The following upper bound holds for Algorithm 3:*

$$\begin{aligned}
\mathbb{E}\sum_{t=0}^{T-1}\sum_{j=1}^{d}\frac{\Delta_{t,j}^2}{\sum_{l=0}^{t}\Delta_{l,j}^2 + \tau^2} \leq \Bigg[\min\Bigg\{ &d + \sum_{j=1}^{d}\log\left(1 + \frac{\eta_l^2 K^2 G^2 T}{\tau^2}\right), \\
&\frac{4\eta_l^2 K T}{m\tau^2}\sum_{j=1}^{d}\sigma_{l,j}^2 + 20\eta_l^4 K^3 L^2 T\mathbb{E}\sum_{j=1}^{d}\frac{(\sigma_{l,j}^2 + 6K\sigma_{g,j}^2)}{\tau^2} + \frac{40\eta_l^4 K^2 L^2}{\tau^2}\sum_{t=0}^{T-1}\mathbb{E}\left[\|\nabla f(x_t)\|^2\right]\Bigg\}\Bigg]
\end{aligned}$$

Proof. We bound the desired quantity in the following manner:

$$\mathbb{E}\sum_{t=0}^{T-1}\sum_{j=1}^{d}\frac{\Delta_{t,j}^2}{\sum_{l=0}^{t}\Delta_{l,j}^2 + \tau^2} \leq d + \mathbb{E}\sum_{j=1}^{d}\log\left(1 + \frac{\sum_{l=0}^{T-1}\Delta_{l,j}^2}{\tau^2}\right) \leq d + \sum_{j=1}^{d}\log\left(1 + \frac{\eta_l^2 K^2 G^2 T}{\tau^2}\right).$$

An alternate way of the bounding this quantity is as follows:

$$\begin{aligned}
&\mathbb{E}\sum_{t=0}^{T-1}\sum_{j=1}^{d}\frac{\Delta_{t,j}^2}{\sum_{l=0}^{t}\Delta_{t,j}^2+\tau^2} \leq \mathbb{E}\sum_{t=0}^{T-1}\sum_{j=1}^{d}\frac{\Delta_{t,j}^2}{\tau^2}\\
&\leq \mathbb{E}\sum_{t=0}^{T-1}\left\|\frac{\Delta_t+\eta_l K\nabla f(x_t)-\eta_l K\nabla f(x_t)}{\tau}\right\|^2\\
&\leq 2\mathbb{E}\sum_{t=0}^{T-1}\left[\left\|\frac{\Delta_t+\eta_l K\nabla f(x_t)}{\tau}\right\|^2+\eta_l^2K^2\left\|\frac{\nabla f(x_t)}{\tau}\right\|^2\right]
\end{aligned}$$

The first quantity in the above bound can be further bounded as follows:

$$\begin{aligned}
&2\mathbb{E}\sum_{t=0}^{T-1}\left\|\frac{1}{\tau}\cdot\left(-\frac{1}{m}\sum_{i=1}^{m}\sum_{k=0}^{K-1}\eta_l g_{i,k}^t+\eta_l K\nabla f(x_t)\right)\right\|^2\\
&=2\mathbb{E}\sum_{t=0}^{T-1}\left[\left\|\frac{1}{\tau}\cdot\left(\frac{1}{m}\sum_{i=1}^{m}\sum_{k=0}^{K-1}\left(\eta_l g_{i,k}^t-\eta_l\nabla F_i(x_{i,k}^t)+\eta_l\nabla F_i(x_{i,k}^t)-\eta_l\nabla F_i(x_t)+\eta_l\nabla F_i(x_t)\right)-\eta_l K\nabla f(x_t)\right)\right\|^2\right]\\
&=\frac{2\eta_l^2}{m^2}\sum_{t=0}^{T-1}\mathbb{E}\left[\left\|\sum_{i=1}^{m}\sum_{k=0}^{K-1}\frac{1}{\tau}\cdot\left(g_{i,k}^t-\nabla F_i(x_{i,k}^t)+\nabla F_i(x_{i,k}^t)-\nabla F_i(x_t)\right)\right\|^2\right]\\
&\leq\frac{4\eta_l^2}{m^2}\sum_{t=0}^{T-1}\mathbb{E}\left[\left\|\sum_{i=1}^{m}\sum_{k=0}^{K-1}\frac{1}{\tau}\cdot\left(g_{i,k}^t-\nabla F_i(x_{i,k}^t)\right)\right\|^2+\left\|\sum_{i=1}^{m}\sum_{k=0}^{K-1}\frac{1}{\tau}\cdot\left(\nabla F_i(x_{i,k}^t)-\nabla F_i(x_t)\right)\right\|^2\right]\\
&\leq\frac{4\eta_l^2KT}{m}\sum_{j=1}^{d}\frac{\sigma_{l,j}^2}{\tau^2}+\frac{4\eta_l^2K}{m}\sum_{i=1}^{m}\sum_{k=0}^{K-1}\sum_{t=0}^{T-1}\left\|\frac{1}{\tau}\cdot\left(\nabla F_i(x_{i,k}^t)-\nabla F_i(x_t)\right)\right\|^2\\
&\leq\frac{4\eta_l^2KT}{m}\sum_{j=1}^{d}\frac{\sigma_{l,j}^2}{\tau^2}+\frac{4\eta_l^2K}{m}\sum_{i=1}^{m}\sum_{k=0}^{K-1}\sum_{t=0}^{T-1}\left\|\frac{L}{\tau}\cdot\left(x_{i,k}^t-x_t\right)\right\|^2 \qquad \text{using Assumption 1 and Assumption 2}\\
&\leq\frac{4\eta_l^2KT}{m\tau^2}\sum_{j=1}^{d}\sigma_{l,j}^2+20\eta_l^4K^3L^2T\mathbb{E}\sum_{j=1}^{d}\frac{(\sigma_{l,j}^2+6K\sigma_{g,j}^2)}{\tau^2}+\frac{40\eta_l^4K^4L^2}{\tau^2}\sum_{t=0}^{T-1}\mathbb{E}\left[\left\|\nabla f(x_t)\right\|^2\right] \qquad \text{using Lemma 4.}
\end{aligned}$$

Here, the first inequality follows from simple application of the fact that $ab \leq (a^2+b^2)/2$. The result follows. □

Substituting the above bound in Equation (10), we obtain:

$$\begin{aligned}
&\frac{\eta_l K}{4}\sum_{t=0}^{T-1}\sum_{j=1}^{d}\mathbb{E}\frac{[\nabla f(x_t)]_j^2}{\sqrt{v_{t-1,j}}+\tau}\\
&\leq\frac{f(x_0)-\mathbb{E}[f(x_T)]}{\eta}+\frac{5\eta_l^3K^2L^2}{2\tau}\mathbb{E}\sum_{t=0}^{T-1}\sum_{j=1}^{d}(\sigma_{l,j}^2+6K\sigma_{g,j}^2)\\
&\qquad+\sum_{j=1}^{d}\left(\frac{\eta_l KG^2}{\tau}+\frac{\eta L}{2}\right)\times\left[\min\left\{d+d\log\left(1+\frac{\eta_l^2K^2G^2T}{\tau^2}\right),\right.\right.\\
&\qquad\qquad\left.\left.\frac{4\eta_l^2KT}{m\tau^2}\sum_{j=1}^{d}\sigma_{l,j}^2+20\eta_l^4K^3L^2T\mathbb{E}\sum_{j=1}^{d}\frac{(\sigma_{l,j}^2+6K\sigma_{g,j}^2)}{\tau^2}+\frac{40\eta_l^4K^4L^2}{\tau^2}\sum_{t=0}^{T-1}\mathbb{E}\left[\left\|\nabla f(x_t)\right\|^2\right]\right\}\right]
\end{aligned}\tag{11}$$

We observe the following:

$$\sum_{t=0}^{T-1}\sum_{j=1}^{d}\frac{[\nabla \mathbb{E} f(x_t)]_j^2}{\sqrt{v_{t-1,j}}+\tau} \geq \sum_{t=0}^{T-1}\sum_{j=1}^{d}\mathbb{E}\frac{[\nabla f(x_t)]_j^2}{\eta_l K G\sqrt{T}+\tau} \geq \frac{T}{\eta_l K G\sqrt{T}+\tau}\min_{0\leq t\leq T}\mathbb{E}\|\nabla f(x_t)\|^2.$$

The second part of Theorem 1 follows from using the above inequality in Equation (11). Note that the first part of Theorem 1 is obtain from the part of Lemma 5. □

A.2 Proof of Theorem 2

Proof of Theorem 2. The proof strategy is similar to that of FEDADAGRAD except that we need to handle the exponential moving average in FEDADAM. We note that the update of FEDADAM is the following

$$x_{t+1} = x_t + \eta\frac{\Delta_t}{\sqrt{v_t}+\tau},$$

for all $i \in [d]$. Using the L-smooth nature of function f and the above update rule, we have the following:

$$f(x_{t+1}) \leq f(x_t) + \eta\left\langle \nabla f(x_t), \frac{\Delta_t}{\sqrt{v_t}+\tau}\right\rangle + \frac{\eta^2 L}{2}\sum_{i=1}^{d}\frac{\Delta_{t,i}^2}{(\sqrt{v_{t,i}}+\tau)^2} \tag{12}$$

The second step follows simply from FEDADAM's update. We take the expectation of $f(x_{t+1})$ (over randomness at time step t) and rewrite the above inequality as:

$$\begin{aligned}
\mathbb{E}_t[f(x_{t+1})] &\leq f(x_t) + \eta\left\langle \nabla f(x_t), \mathbb{E}_t\left[\frac{\Delta_t}{\sqrt{v_t}+\tau} - \frac{\Delta_t}{\sqrt{\beta_2 v_{t-1}}+\tau} + \frac{\Delta_t}{\sqrt{\beta_2 v_{t-1}}+\tau}\right]\right\rangle + \frac{\eta^2 L}{2}\sum_{j=1}^{d}\mathbb{E}_t\left[\frac{\Delta_{t,j}^2}{(\sqrt{v_{t,j}}+\tau)^2}\right] \\
&= f(x_t) + \underbrace{\eta\left\langle \nabla f(x_t), \mathbb{E}_t\left[\frac{\Delta_t}{\sqrt{\beta_2 v_{t-1}}+\tau}\right]\right\rangle}_{R_1} + \underbrace{\eta\left\langle \nabla f(x_t), \mathbb{E}_t\left[\frac{\Delta_t}{\sqrt{v_t}+\tau} - \frac{\Delta_t}{\sqrt{\beta_2 v_{t-1}}+\tau}\right]\right\rangle}_{R_2} \\
&\quad + \frac{\eta^2 L}{2}\sum_{j=1}^{d}\mathbb{E}_t\left[\frac{\Delta_{t,j}^2}{(\sqrt{v_{t,j}}+\tau)^2}\right]
\end{aligned} \tag{13}$$

Bounding R_2. We observe the following about R_2:

$$\begin{aligned}
R_2 &= \mathbb{E}_t\sum_{j=1}^{d}[\nabla f(x_t)]_j \times \left[\frac{\Delta_{t,j}}{\sqrt{v_{t,j}}+\tau} - \frac{\Delta_{t,j}}{\sqrt{\beta_2 v_{t-1,j}}+\tau}\right] \\
&= \mathbb{E}_t\sum_{j=1}^{d}[\nabla f(x_t)]_j \times \Delta_{t,j} \times \left[\frac{\sqrt{\beta_2 v_{t-1,j}} - \sqrt{v_{t,j}}}{(\sqrt{v_{t,j}}+\tau)(\sqrt{\beta_2 v_{t-1,j}}+\tau)}\right] \\
&= \mathbb{E}_t\sum_{j=1}^{d}[\nabla f(x_t)]_j \times \Delta_{t,j} \times \left[\frac{-(1-\beta_2)\Delta_{t,j}^2}{(\sqrt{v_{t,j}}+\tau)(\sqrt{\beta_2 v_{t-1,j}}+\tau)(\sqrt{\beta_2 v_{t-1,j}}+\sqrt{v_{t,j}})}\right] \\
&\leq (1-\beta_2)\mathbb{E}_t\sum_{j=1}^{d}|[\nabla f(x_t)]_j| \times |\Delta_{t,j}| \times \left[\frac{\Delta_{t,j}^2}{(\sqrt{v_{t,j}}+\tau)(\sqrt{\beta_2 v_{t-1,j}}+\tau)(\sqrt{\beta_2 v_{t-1,j}}+\sqrt{v_{t,j}})}\right] \\
&\leq \sqrt{1-\beta_2}\mathbb{E}_t\sum_{j=1}^{d}|[\nabla f(x_t)]_j| \times \left[\frac{\Delta_{t,j}^2}{(\sqrt{v_{t,j}}+\tau)(\sqrt{\beta_2 v_{t-1,j}}+\tau)}\right] \\
&\leq \sqrt{1-\beta_2}\mathbb{E}_t\sum_{j=1}^{d}\frac{G}{\tau} \times \left[\frac{\Delta_{t,j}^2}{\sqrt{v_{t,j}}+\tau}\right].
\end{aligned}$$

Bounding R_1. The term R_1 can be bounded as follows:

$$\begin{aligned}
R_1 &= \left\langle \nabla f(x_t), \mathbb{E}_t \left[\frac{\Delta_t}{\sqrt{\beta_2 v_{t-1}} + \tau} \right] \right\rangle \\
&= \left\langle \frac{\nabla f(x_t)}{\sqrt{\beta_2 v_{t-1}} + \tau}, \mathbb{E}_t \left[\Delta_t - \eta_l K \nabla f(x_t) + \eta_l K \nabla f(x_t) \right] \right\rangle \\
&= -\eta_l K \sum_{j=1}^{d} \frac{[\nabla f(x_t)]_j^2}{\sqrt{\beta_2 v_{t-1,j}} + \tau} + \underbrace{\left\langle \frac{\nabla f(x_t)}{\sqrt{\beta_2 v_{t-1}} + \tau}, \mathbb{E}_t \left[\Delta_t + \eta_l K \nabla f(x_t) \right] \right\rangle}_{R_3}.
\end{aligned} \tag{14}$$

Bounding R_3. The term R_3 can be bounded in exactly the same way as term T_3 in proof of Theorem 1:

$$R_3 \leq \frac{\eta_l K}{2} \sum_{j=1}^{d} \frac{[\nabla f(x_t)]_j^2}{\sqrt{\beta_2 v_{t-1,j}} + \tau} + \frac{\eta_l L^2}{2m\tau} \mathbb{E}_t \left[\sum_{i=1}^{m} \sum_{k=0}^{K-1} \|x_{i,k}^t - x_t\|^2 \right]$$

Substituting the above inequality in Equation (14), we get

$$R_1 \leq -\frac{\eta_l K}{2} \sum_{j=1}^{d} \frac{[\nabla f(x_t)]_j^2}{\sqrt{\beta_2 v_{t-1,j}} + \tau} + \frac{\eta_l L^2}{2m\tau} \mathbb{E}_t \left[\sum_{i=1}^{m} \sum_{k=0}^{K-1} \|x_{i,k}^t - x_t\|^2 \right]$$

Using Lemma 4, we obtain the following bound on R_1:

$$R_1 \leq -\frac{\eta_l K}{4} \sum_{j=1}^{d} \frac{[\nabla f(x_t)]_j^2}{\sqrt{\beta_2 v_{t-1,j}} + \tau} + \frac{5\eta_l^3 K^2 L^2}{2\tau} \mathbb{E}_t \sum_{j=1}^{d} (\sigma_{l,j}^2 + 6K\sigma_{g,j}^2) \tag{15}$$

Putting pieces together. Substituting bounds R_1 and R_2 in Equation (13), we have

$$\begin{aligned}
\mathbb{E}_t[f(x_{t+1})] \leq f(x_t) &- \frac{\eta\eta_l K}{4} \sum_{j=1}^{d} \frac{[\nabla f(x_t)]_j^2}{\sqrt{\beta_2 v_{t-1,j}} + \tau} + \frac{5\eta\eta_l^3 K^2 L^2}{2\tau} \mathbb{E} \sum_{j=1}^{d} (\sigma_{l,j}^2 + 6K\sigma_{g,j}^2) \\
&+ \left(\frac{\eta\sqrt{1-\beta_2} G}{\tau} \right) \sum_{j=1}^{d} \mathbb{E}_t \left[\frac{\Delta_{t,j}^2}{\sqrt{v_{t,j}} + \tau)} \right] + \left(\frac{\eta^2 L}{2} \right) \sum_{j=1}^{d} \mathbb{E}_t \left[\frac{\Delta_{t,j}^2}{v_{t,j} + \tau^2)} \right]
\end{aligned}$$

Summing over $t = 0$ to $T-1$ and using telescoping sum, we have

$$\begin{aligned}
\mathbb{E}[f(x_T)] \leq f(x_0) &- \frac{\eta\eta_l K}{4} \sum_{t=0}^{T-1} \sum_{j=1}^{d} \mathbb{E} \frac{[\nabla f(x_t)]_j^2}{\sqrt{\beta_2 v_{t-1,j}} + \tau} + \frac{5\eta\eta_l^3 K^2 L^2 T}{2\tau} \mathbb{E} \sum_{j=1}^{d} (\sigma_{l,j}^2 + 6K\sigma_{g,j}^2) \\
&\left(\frac{\eta\sqrt{1-\beta_2} G}{\tau} \right) \sum_{t=0}^{T-1} \sum_{j=1}^{d} \mathbb{E} \left[\frac{\Delta_{t,j}^2}{\sqrt{v_{t,j}} + \tau)} \right] + \left(\frac{\eta^2 L}{2} \right) \sum_{t=0}^{T-1} \sum_{j=1}^{d} \mathbb{E} \left[\frac{\Delta_{t,j}^2}{v_{t,j} + \tau^2)} \right]
\end{aligned} \tag{16}$$

To bound this term further, we need the following result.

Lemma 6. *The following upper bound holds for Algorithm 4:*

$$\sum_{t=0}^{T-1} \sum_{j=1}^{d} \mathbb{E} \left[\frac{\Delta_{t,j}^2}{(v_{t,j} + \tau^2)} \right] \leq \frac{4\eta_l^2 KT}{m\tau^2} \sum_{j=1}^{d} \sigma_{l,j}^2 + \frac{20\eta_l^4 K^3 L^2 T}{\tau^2} \mathbb{E} \sum_{j=1}^{d} (\sigma_{l,j}^2 + 6K\sigma_{g,j}^2) + \frac{40\eta_l^4 K^4 L^2}{\tau^2} \sum_{t=0}^{T-1} \mathbb{E} \left[\|\nabla f(x_t)\|^2 \right]$$

Proof.

$$\mathbb{E}\sum_{t=0}^{T-1}\sum_{j=1}^{d}\frac{\Delta_{t,j}^2}{(1-\beta_2)\sum_{l=0}^{t}\beta_2^{t-l}\Delta_{t,j}^2+\tau^2}\leq\mathbb{E}\sum_{t=0}^{T-1}\sum_{j=1}^{d}\frac{\Delta_{t,j}^2}{\tau^2}$$

The rest of the proof follows along the lines of proof of Lemma 5. Using the same argument, we get

$$\sum_{j=1}^{d}\mathbb{E}\left[\frac{\Delta_{t,j}^2}{(v_{t,j}+\tau^2)}\right]\leq\frac{4\eta_l^2KT}{m\tau^2}\sum_{j=1}^{d}\sigma_{l,j}^2+\frac{20\eta_l^4K^3L^2T}{\tau^2}\mathbb{E}\sum_{j=1}^{d}(\sigma_{l,j}^2+6K\sigma_{g,j}^2)$$
$$+\frac{40\eta_l^4K^4L^2}{\tau^2}\sum_{t=0}^{T-1}\mathbb{E}\left[\|\nabla f(x_t)\|^2\right],$$

which is the desired result. □

Substituting the bound obtained from above lemma in Equation (16) and using a similar argument for bounding

$$\left(\frac{\eta\sqrt{1-\beta_2}G}{\tau}\right)\sum_{t=0}^{T-1}\sum_{j=1}^{d}\mathbb{E}\left[\frac{\Delta_{t,j}^2}{\sqrt{v_{t,j}}+\tau)}\right],$$

we obtain

$$\mathbb{E}_t[f(x_T)]\leq f(x_0)-\frac{\eta\eta_lK}{8}\sum_{t=0}^{T-1}\sum_{j=1}^{d}\frac{[\nabla f(x_t)]_j^2}{\sqrt{\beta_2v_{t-1,j}}+\tau}+\frac{5\eta\eta_l^3K^2L^2T}{2\tau}\mathbb{E}\sum_{j=1}^{d}(\sigma_{l,j}^2+6K\sigma_{g,j}^2)$$
$$+\left(\eta\sqrt{1-\beta_2}G+\frac{\eta^2L}{2}\right)\times\left[\frac{4\eta_l^2KT}{m\tau^2}\sum_{j=1}^{d}\sigma_{l,j}^2+\frac{20\eta_l^4K^4L^2T}{\tau^2}\mathbb{E}\sum_{j=1}^{d}(\sigma_{l,j}^2+6K\sigma_{g,j}^2)\right]$$

The above inequality is obtained due to the fact that:

$$\left(\sqrt{1-\beta_2}G+\frac{\eta L}{2}\right)\frac{40\eta_l^4K^4L^2}{\tau^2}\leq\frac{\eta_lK}{16}\left(\frac{1}{\sqrt{\beta_2}\eta_lKG}+\frac{1}{\tau}\right).$$

The above condition follows from the condition on η_l in Theorem 2. We also observe the following:

$$\sum_{t=0}^{T-1}\sum_{j=1}^{d}\frac{[\nabla f(x_t)]_j^2}{\sqrt{\beta_2v_{t-1,j}}+\tau}\geq\sum_{t=0}^{T-1}\sum_{j=1}^{d}\frac{[\nabla f(x_t)]_j^2}{\sqrt{\beta_2}\eta_lKG+\tau}\geq\frac{T}{\sqrt{\beta_2}\eta_lKG+\tau}\min_{0\leq t\leq T}\|\nabla f(x_t)\|^2.$$

Substituting this bound in the above equation yields the desired result. □

A.3 Auxiliary Lemmatta

Lemma 7. *For random variables $z_1,\ldots,z_r$, we have*

$$\mathbb{E}\left[\|z_1+\ldots+z_r\|^2\right]\leq r\mathbb{E}\left[\|z_1\|^2+\ldots+\|z_r\|^2\right].$$

Lemma 8. *For random variables $z_1,\ldots,z_r$ are independent and mean 0, we have*

$$\mathbb{E}\left[\|z_1+\ldots+z_r\|^2\right]=\mathbb{E}\left[\|z_1\|^2+\ldots+\|z_r\|^2\right].$$

B Dataset & Models

Here we provide detailed description of the datasets and models used in the paper. We use federated versions of vision datasets EMNIST (Cohen et al., 2017) and CIFAR-100 (Krizhevsky and Hinton, 2009), and language modeling datasets Shakespeare (McMahan et al., 2017) and StackOverflow[3]. Statistics for the training datasets can be found in Table 1. We give descriptions of the datasets, models, and tasks below.

B.1 CIFAR-100

We introduce a federated version of CIFAR-100 by randomly partitioning the training data among 500 clients, with each client receiving 100 examples. While Hsu et al. (2019) used latent Dirichlet allocation (LDA) over the fine labels of CIFAR-100 to create a federated dataset, we use a two step LDA process over the coarse and fine labels. We randomly partition the data to reflect the "coarse" and "fine" label structure of CIFAR-100 by using Pachinko Allocation (Li and McCallum, 2006). This creates more realistic client datasets, whose label distributions better resemble practical heterogeneous client datasets. We train a modified ResNet-18 on this dataset, where the batch normalization layers are replaced by group normalization layers (Wu and He, 2018), with two groups. As shown by Hsieh et al. (2019), group normalization can lead to significant gains in accuracy over batch normalization in federated settings.

Creating a federated CIFAR-100 As discussed above, we use the Pachinko Allocation Method (PAM) introduced by Li and McCallum (2006) to create a federated CIFAR-100. PAM is a topic modeling method in which the correlations between individual words in a vocabulary are represented by a rooted directed acyclic graph (DAG) whose leaves are the vocabulary words. The interior nodes are topics which have a corresponding Dirichlet distribution over their child nodes. To generate a document, we first sample a multinomial distributions from each interior node's Dirichlet distribution. To sample a word from the document, we begin at the root, and draw a sample over its children from its multinomial distribution. We continue sampling child nodes according to these multinomial distributions until we reach a leaf node.

To partition CIFAR-100 across clients, we use the coarse-fine label structure of CIFAR-100. Each image in the dataset has a fine label (often referred to as just its label) which is a member of a more general coarse label. For example, an image may have the fine label "seal", which is a member of the coarse label "aquatic mammals". There are 20 coarse labels in CIFAR-100, each with 5 associated fine labels. We represent this structure as a DAG G, with a single root whose children are the coarse labels. Each coarse label is an interior node of the DAG whose child nodes are its associated fine labels. For example, the coarse label "flowers" has child nodes "orchids", "poppies", "roses", "sunflowers", and "tulips". The root node has an associated symmetric Dirichlet distribution with parameter $\alpha = 0.1$ over the coarse labels, and each coarse label has a symmetric Dirichlet distribution with parameter $\beta = 10.0$.

To create clients, we associate each client to a document. That is, we draw a multinomial distribution from the Dirichlet prior at the root $(\text{Dir}(\alpha))$ and a multinomial distribution from the Dirichlet prior at each coarse label $(\text{Dir}(\beta))$. To create the client dataset, we draw samples from this DAG using Pachinko allocation. This gives us some fine label leaf node. We then draw a random sample from CIFAR-100 with the given fine label, and assign it to the client's dataset. We do this 100 times for 500 distinct clients.

While more complicated than simply using LDA over the labels, this approach creates more realistic heterogeneity among client datasets. In particular, this leads to correlations between label frequencies for fine labels within the same coarse label set. Intuitively, if a client corresponds to an edge device with many pictures of dolphins, they are more likely to also have pictures of whales. We set our dirichlet parameters α and β to reflect this; By using a small α at the root, client datsets become more likely to focus on a few coarse labels. By using a larger β for the coarse-to-fine label distributions, we increase the likelihood that the clients have multiple fine labels from the same coarse label.

There is one important nuance in the above. Once we sample a fine label using PAM, we then randomly select an element of CIFAR-100 with the same fine label, but *without replacement*. This ensure that no two clients have overlapping examples. To do this, we generate client datasets in order. Suppose that we have just sampled fine label y with coarse

[3] https://archive.org/download/stackexchange

label c for client m. Further suppose that there is only one remaining example (x, c, y) in CIFAR-100 with image x, coarse label c and fine label y that has not already been allocated to a client. We assign (x, c, y) to client m's dataset. We then remove the leaf node y from the DAG G used with PAM, so that future clients cannot sample y. We must remove outcome y from the multinomial distribution θ_c that client m has associated to coarse label c, which we refer to as *renormalization* with respect to y (see Algorithm 6). If i has no remaining children after pruning node j, we also remove node i from the graph and re-normalize the root multinomial distribution θ_r with respect to c. For all subsequent clients, we draw multinomials from this pruned G according to symmetric Dirichlet distributions with the same parameters as before, but with one fewer category.

Algorithm 5 Creating a federated CIFAR-100

Initialization: $N, M \in \mathbb{Z}_{>0}, \alpha, \beta \in \mathbb{R}_{\geq 0}$
for $m = 1, \cdots, M$ **do**
  Sample $\theta_r \sim \text{Dir}(\alpha, |G[r]|)$
  for $c \in \mathcal{C} \cap G[r]$ **do**
    Sample $\theta_c \sim \text{Dir}(\beta, |G[c]|)$
  $D_m = \emptyset$
  for $n = 1, \cdots N$ **do**
    Sample $c \sim \text{Multinomial}(\theta_r)$
    Sample $y \sim \text{Multinomial}(\theta_c)$
    Select $(x, c, y) \in \mathcal{S}$ uniformly at random
    $D_m = D_m \cup \{(x, c, y)\}$
    $\mathcal{S} = \mathcal{S} \backslash \{(x, c, y)\}$
    if $\mathcal{S}_y = \emptyset$ **then**
      $G = G \backslash \{y\}$
      $\theta_c = \text{RENORMALIZE}(\theta_c, y)$
      if $\mathcal{S}_c = \emptyset$ **then**
        $G = G \backslash \{c\}$
        $\theta_r = \text{RENORMALIZE}(\theta_r, c)$

Algorithm 6 RENORMALIZE

Initialization: $\theta = (p_1, \ldots, p_K), i \in [K]$
$a = \sum_{k \neq i} p_k$
for $k \in [K], k \neq i$ **do**
  $p'_k = \dfrac{p_k}{a}$
Return $\theta' = (p'_1, \cdots p'_{i-1}, p'_{i+1}, \cdots, p'_K)$

We now introduce notation for a full version of our algorithm. Let $\mathcal{C}$ denote the set of fine labels and $\mathcal{Y}$ the set of fine labels. We let $\mathcal{S}$ denote the overall CIFAR-100 dataset, consisting of examples of the form (x, c, y) where x is an image vector, $c \in \mathcal{C}$ is a coarse label set, and $y \in \mathcal{Y}$ is a fine label satisfying $y \in c$. Given $c \in \mathcal{C}, y \in \mathcal{Y}$, we let $\mathcal{S}_c$ and $\mathcal{S}_y$ denote the set of examples in $\mathcal{S}$ with coarse label c and fine label y (respectively). Given a node $v \in G$ (which can either be the root r, a coarse label, or a fine label), we let $|G[v]|$ denote the set of children of v in G. For $\gamma \in \mathbb{R}$, we let $\text{Dir}(\gamma, k)$ denote the symmetric Dirichlet distribution with k categories. We let M denote the number of clients, N denote the number of examples per client, and D_m denote the dataset for client $m \in \{1, \cdots, M\}$. A full description of our method for creating a federated CIFAR-100, using this notation, is given in Algorithm 5. For our experiments, we use $N = 100, M = 500, \alpha = 0.1, \beta = 10$.

In Figure 5, we plot the distribution of unique labels among the 500 training clients. We see that each client has only a fraction of the overall labels in the distribution. Moreover, there is some variance to the number of unique clients, with most having between 20 and 30, and some having over 40. Other client datasets have very small numbers of unique labels. While this is primarily an artifact of performing without replacement sampling, this helps increase the heterogeneity of the dataset in a way that reflects practical concerns. In many settings, clients may only have a few types of labels in their datset.

Preprocessing CIFAR-100 consists of images with 3 channels of 32×32 pixels each. Each pixel is represented by an unsigned int8. As is standard with CIFAR-100, we perform preprocessing on both training and test images. For training images, we perform a random crop to shape $(24, 24, 3)$, followed by a random horizontal flip. For testing images, we centrally crop the image to $(24, 24, 3)$. For both training and testing images, we then normalize the pixel

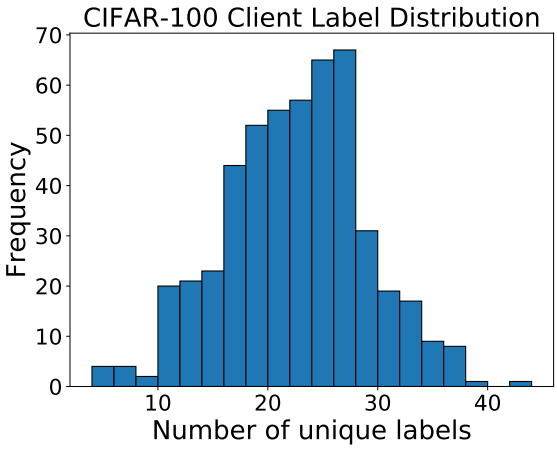

Figure 5: The distribution of the number of unique labels among client datasets in our federated CIFAR-100 dataset.

values according to their mean and standard deviation. Namely, given an image x, we compute $(x - \mu)/\sigma$ where μ is the average of the pixel values in x, and σ is the standard deviation.

B.2 EMNIST

EMNIST consists of images of digits and upper and lower case English characters, with 62 total classes. The federated version of EMNIST (Caldas et al., 2018) partitions the digits by their author. The dataset has natural heterogeneity stemming from the writing style of each person. We perform two distinct tasks on EMNIST, autoencoder training (EMNIST AE) and character recognition (EMNIST CR). For EMNIST AE, we train the "MNIST" autoencoder (Zaheer et al., 2018). This is a densely connected autoencoder with layers of size $(28 \times 28) - 1000 - 500 - 250 - 30$ and a symmetric decoder. A full description of the model is in Table 4. For EMNIST CR, we use a convolutional network. The network has two convolutional layers (with 3×3 kernels), max pooling, and dropout, followed by a 128 unit dense layer. A full description of the model is in Table 5.

Table 4: EMNIST autoencoder model architecture. We use a sigmoid activation at all dense layers.

Layer	Output Shape	# of Trainable Parameters
Input	784	0
Dense	1000	785000
Dense	500	500500
Dense	250	125250
Dense	30	7530
Dense	250	7750
Dense	500	125500
Dense	1000	501000
Dense	784	784784

Table 5: EMNIST character recognition model architecture.

Layer	Output Shape	# of Trainable Parameters	Activation	Hyperparameters
Input	$(28, 28, 1)$	0		
Conv2d	$(26, 26, 32)$	320		kernel size = 3; strides=$(1, 1)$
Conv2d	$(24, 24, 64)$	18496	ReLU	kernel size = 3; strides=$(1, 1)$
MaxPool2d	$(12, 12, 64)$	0		pool size= $(2, 2)$
Dropout	$(12, 12, 64)$	0		$p = 0.25$
Flatten	9216	0		
Dense	128	1179776		
Dropout	128	0		$p = 0.5$
Dense	62	7998	softmax	

B.3 Shakespeare

Shakespeare is a language modeling dataset built from the collective works of William Shakespeare. In this dataset, each client corresponds to a speaking role with at least two lines. The dataset consists of 715 clients. Each client's lines are partitioned into training and test sets. Here, the task is to do next character prediction. We use an RNN that first takes a series of characters as input and embeds each of them into a learned 8-dimensional space. The embedded characters are then passed through 2 LSTM layers, each with 256 nodes, followed by a densely connected softmax output layer. We split the lines of each speaking role into into sequences of 80 characters, padding if necessary. We use a vocabulary size of 90; 86 for the characters contained in the Shakespeare dataset, and 4 extra characters for padding, out-of-vocabular, beginning of line and end of line tokens. We train our model to take a sequence of 80 characters, and predict a sequence of 80 characters formed by shifting the input sequence by one (so that its last character is the new character we are actually trying to predict). Therefore, our output dimension is 80×90. A full description of the model is in Table 6.

Table 6: Shakespeare model architecture.

Layer	Output Shape	# of Trainable Parameters
Input	80	0
Embedding	$(80, 8)$	720
LSTM	$(80, 256)$	271360
LSTM	$(80, 256)$	525312
Dense	$(80, 90)$	23130

B.4 StackOverflow

StackOverflow is a language modeling dataset consisting of question and answers from the question and answer site, StackOverflow. The questions and answers also have associated metadata, including tags. The dataset contains 342,477 unique users which we use as clients. We perform two tasks on this dataset: tag prediction via logistic regression (StackOverflow LR), and next-word prediction (StackOverflow NWP). For both tasks, we restrict to the 10,000 most frequently used words. For StackOverflow LR and NWP, we restrict to at most 500 and 128 sentences per user (respectively) for the client datasets. For StackOverflow LR, we restrict to the 500 most frequent tags and adopt a one-versus-rest classification strategy, where each question/answer is represented as a bag-of-words vector (normalized to have sum 1). For StackOverflow NWP, we perform padding and truncation to ensure that sentences have 20 words,

and then represent the sentence as a sequence of indices corresponding to the 10,000 frequently used words, as well as indices representing padding, out-of-vocabulary words, beginning of sentence, and end of sentence. We perform next word prediction on these sequences using an RNN that embeds each word in a sentence into a learned 96-dimensional space. It then feeds the embedded words into a single LSTM layer of hidden dimension 670, followed by a densely connected softmax output layer. A full description of the model is in Table 7. The metrics reported in the main body are top-1 accuracy over the proper 10,000-word vocabulary; that is, they do not include padding, out-of-vocab, or beginning or end of sentence tokens.

Table 7: StackOverflow next word prediction model architecture.

Layer	Output Shape	# of Trainable Parameters
Input	20	0
Embedding	$(20, 96)$	960384
LSTM	$(20, 670)$	2055560
Dense	$(20, 96)$	64416
Dense	$(20, 10004)$	970388

C Experiment hyperparameters

C.1 Hyperparameter tuning

Throughout our experiments, we compare the performance of different instantiations of Algorithm 2 that use different server optimizers. We use SGD, SGD with momentum (denoted SGDM), ADAGRAD, ADAM, and YOGI. For the client optimizer, we use mini-batch SGD throughout. For all tasks, we initially tune the client learning rate η_l and server learning rate η over the grids $\{10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 1\}$ and $\{10^{-3}, 10^{-2}, 10^{-1}, 1, 10\}$ respectively. For EMNIST and Shakespeare, we then refine the grid based on an initial round of experiments in order to provide more accurate results. For CIFAR-100 and StackOverflow NWP, we do no further tuning due to the relatively long training times. When comparing our algorithms to FEDAVG, we use the same client learning rate grid used for the other optimizers, and set $\eta = 1$ (as is implicitly done in Algorithm 1). Full descriptions of the per-task server and client learning rate grids are given in Appendix C.2

Given the large number of hyperparameters to tune with the aforementioned methods, and to avoid conflating variables, we fix the batch size at a per-task level. We use a batch size of 20 for CIFAR-100 and EMNIST, a batch size of 4 for Shakespeare (due to the small number of examples available at some clients), and a batch size of 16 for StackOverflow.

We fix momentum terms of 0.9 throughout for FEDAVGM and the adaptive optimizers. For the adaptive optimizers, we fix the numerical stability constant ϵ to be 0.01. This is a larger value than is typically used in centralized training. However, we found that due to the heterogeneity of the client datasets, especially StackOverflow, larger values of ϵ generally stabilized the convergence of adaptive optimizers. For FEDADAM and FEDYOGI, we fix $\beta_2 = 0.99$, while for FEDADAGRAD, we set the initial accumulator to 0.1. For FEDYOGI, we set the initial accumulator according to the strategy outlined by Zaheer et al. (2018): we use the average of the squares of the coordinates of the gradient, where the gradient is evaluated at the initial model, averaged over multiple batches. A full description of the initialization is given in Appendix C.5.

C.2 Client and server learning rate grids

Below, we give the client learning rate (η_l in Algorithm 2) and server learning rate (η in Algorithm 2) grids used for each task.

CIFAR-100:

$$\eta_l \in \{10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 1\}$$
$$\eta \in \{10^{-3}, 10^{-2}, 10^{-1}, 1, 10\}$$

EMNIST AE:

$$\eta_1 \in \{0.01, 0.05, 0.1, 0.5, 1, 5, 10\}$$
$$\eta \in \{10^{-3}, 10^{-2}, 10^{-1}, 1, 10\}$$

EMNIST CR:

$$\eta_l \in \{10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 1\}$$
$$\eta \in \{10^{-3}, 10^{-2.5}, 10^{-2}, 10^{-1.5}, 10^{-1}, 10^{-0.5}, 1, 10^{0.5}, 10\}$$

Shakespeare:

$$\eta_l \in \{10^{-2.5}, 10^{-2}, 10^{-1.5}, 10^{-1}, 10^{-0.5}, 1, 10^{0.5}, 10\}$$
$$\eta \in \{10^{-2}, 10^{-1.5}, 10^{-1}, 10^{-0.5}, 1, 10^{0.5}, 10\}$$

StackOverflow LR:

$$\eta_l \in \{0.01, 0.05, 0.1, 0.5, 1, 5, 10\}$$
$$\eta \in \{10^{-3}, 10^{-2}, 10^{-1}, 1, 10\}$$

StackOverflow NWP:

$$\eta_l \in \{10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 1\}$$
$$\eta \in \{10^{-3}, 10^{-2}, 10^{-1}, 1, 10\}$$

C.3 Per-task batch sizes

We fix the batch sizes used in our experiments at a per-task level. When comparing centralized training to federated training in Section 5.2, we use the same batch size in both federated and centralized training. A full summary of the batch sizes is given in Table 8.

Table 8: Client batch sizes used for each task

TASK	BATCH SIZE
CIFAR-100	20
EMNIST AE	20
EMNIST CR	20
SHAKESPEARE	4
STACKOVERFLOW LR	100
STACKOVERFLOW NWP	16

C.4 Learning rates used in Section 5.1 and 5.2

In this section, we present, for each optimizer, the best client and server learning rates found for the tasks discussed in Section 5.1. Specifically, these are the learning rates used in Figures 1 and 2, as well as Table 2. We also give the optimal learning rates hyperparameters for the experiments in Table 3. The accuracies in Table 2 are obtained using the learning rates in Tables 9 and 10. The accuracies in Table 3 are obtained using the learning rates in Table 11.

Table 9: The client learning rate (η_l) and server learning rate (η) that achieve the accuracies from Table 2. See Appendix C.2 for a full description of the grids.

	FedAdagrad		FedAdam		FedYogi		FedAvgM		FedAvg	
	η_l	η	η_l	η	η_l	η	η_l	η	η_l	η
StackOverflow LR	0.5	10	0.5	1	5	1	1	10	5	1
EMNIST AE	10	0.1	5	0.1	5	0.1	1	1	5	1

Table 10: The base-10 logarithm of the client (η_l) and server (η) learning rate combinations that achieve the accuracies from Table 2. See Appendix C.2 for a full description of the grids.

	FedAdagrad		FedAdam		FedYogi		FedAvgM		FedAvg	
	η_l	η	η_l	η	η_l	η	η_l	η	η_l	η
CIFAR-100	-1	0	-1	-1	-1	-1	-1	0	-1	0
EMNIST CR	-1	-½	-3	0	-3	½	-2	½	-1	0
Shakespeare	0	-½	½	-2	½	-2	½	-3½	0	0
StackOverflow NWP	-1	0	-1	1	-1	1	-1	0	-1	0

Table 11: The base-10 logarithm of the client (η_l) and server (η) learning rate combinations that achieve the accuracies from Table 3 on the EMNIST character recognition task.

	FedAdagrad		FedAdam		FedYogi		FedAvgM		FedAvg	
	η_l	η	η_l	η	η_l	η	η_l	η	η_l	η
Constant	-1	-½	-3	0	-3	½	-2	½	-1	0
InvSqrt	-1	0	-2	0	-2	0	-2	½	-1	½
ExpDecay	-2	0	-3	0	-2	-½	-2	0	-2	½

C.5 Finding initial accumulator values for FedYogi

In the course of our experiments, we found that Yogi was sensitive to the choice of initial accumulator value. Mirroring Zaheer et al. (2018), we set the initial accumulator value per dataset, model, and loss function triplet. We do this by estimating the average of the square value of the gradient of the loss function, evlauated at the initial model point. Formally, let $f(w; z)$ denote the loss of a model $w \in \mathbb{R}^d$ at a sample z. Let w_0 be the initial value of the model. We then

sample multiple batches $B_1, \ldots, B_k$ of data from the overall dataset. To set the initial accumulator value, we compute

$$\zeta = \frac{1}{k} \sum_{i=1}^{k} \frac{1}{|B_i|} \sum_{z \in B_i} \frac{1}{d} \|\nabla f(w_0; z)\|^2.$$

Recall that $\|\cdot\|$ refers to the ℓ_2 norm. We use ζ as the initial accumulator value of YOGI for the given model, dataset, and loss function combination. To sample these batches in federated learning, we select 1% of the clients uniformly at random, without replacement. For each client selected, we sample a random batch from the client's dataset uniformly at random, without replacement. Note that the batch size used is the same as the client batch size used during federated training. Finally, we note that for StackOverflow, we only select 0.1% of the clients, due to the large number of clients.

D Additional Experimental Results

D.1 Comparing client learning rate schedules on EMNIST CR

In Figure 6, we compare the performance of adaptive optimizers on the EMNIST character recognition task using two different types of client learning rate decay schedules: an inverse square root decay, and an exponential decay where the client learning rate decays by a factor of 0.1 every 500 rounds. We also compare to a constant learniing rate. Our results show that while both learning rate schedules can offer improvement over constant learning rates, exponential decay tends to better across all optimizers.

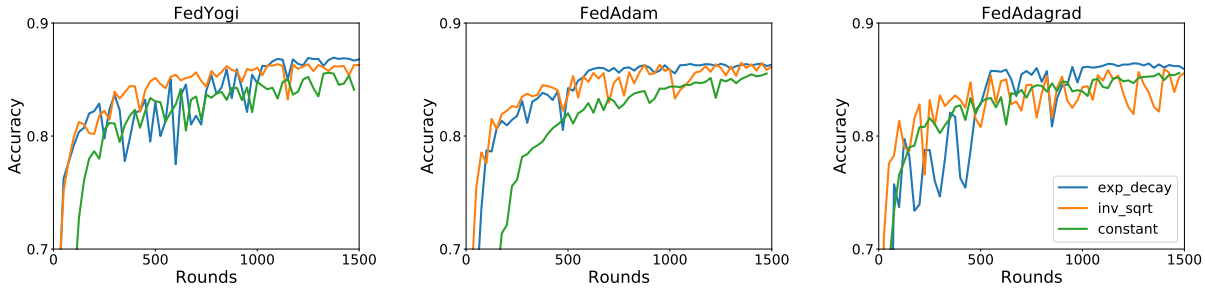

Figure 6: Comparing exponential decay and inverse square root decay client learning rate schedules to constant client learning rates. We plot the accuracy of adaptive server optimizers with the aforementioned client learning rate scheduless on the EMNIST character recognition task. In all settings, we select 10 clients per round and perform 10 local client epochs.

In Figure 7, we compare the accuracy of different federated optimizers with a staircase exponential decay client learning rate schedule. We also plot the maximum accuracy up to a given round for presentation purposes. We find that all optimizers benefit from the exponential decay schedule when we tune the learning rate. However, if we were to simply adapt FEDAVG with a client learning rate schedule (fixing a server learning rate of $\eta = 1$), the resulting algorithm does significantly worse than the adaptive optimizers. It also does worse than FEDAVG on the server with a tuned learning rate of $\eta = \sqrt{10}$. While FEDAVG with a constant server learning rate of $\eta = 1$ may be sufficient for some tasks, client learning rate schedules increase the need for server learning rate tuning.

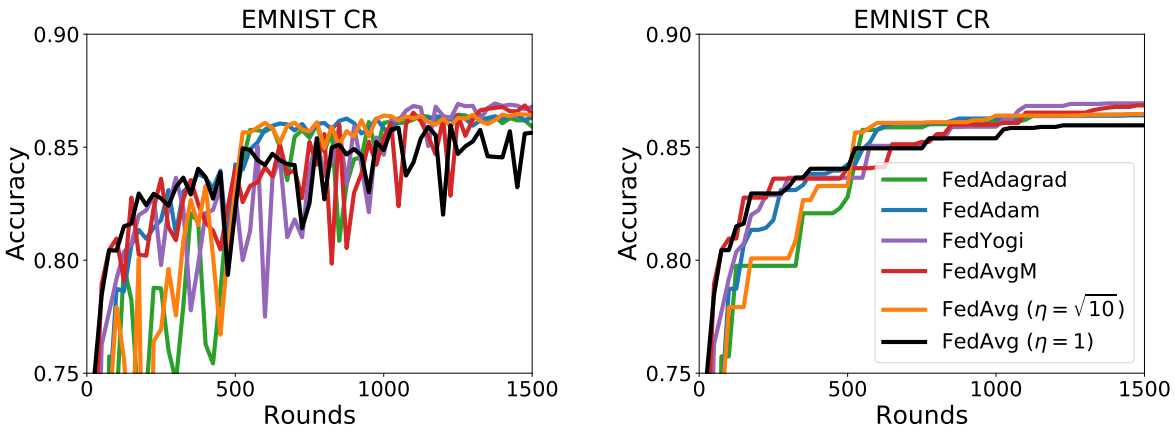

Figure 7: Accuracy of different optimizers on the EMNIST classification task using a staircase exponential decay schedule for the client learning rates. On the left we plot accuracy at a given round, while on the right we plot the maximum accuracy up to a given round. We use 10 local client epochs and 10 clients per round