On Large-Cohort Training for Federated Learning

Anonymous Author(s)
Affiliation
Address
email

Abstract

Federated learning methods typically learn a model by iteratively sampling updates from a population of clients. In this work, we explore how the number of clients sampled at each round (the *cohort size*) impacts the quality of the learned model and the training dynamics of federated learning algorithms. Our work poses three fundamental questions. First, what challenges arise when trying to scale federated learning to larger cohorts? Second, what parallels exist between cohort sizes in federated learning, and batch sizes in centralized learning? Last, how can we design federated learning methods that effectively utilize larger cohort sizes? We give partial answers to these questions based on extensive empirical evaluation. Our work highlights a number of challenges stemming from the use of larger cohorts. While some of these (such as generalization issues and diminishing returns) are analogs of large-batch training challenges, others (including catastrophic training failures and fairness concerns) are unique to federated learning.

1 Introduction

Federated learning (FL) [51] considers learning a model from multiple clients without directly sharing training data, often under the orchestration of a central server. In this work we focus on *cross-device* FL, in which the aim is to learn across a large population of edge devices [27, Table 1]. A distinguishing characteristic of cross-device FL is *partial participation* of the client population: Due to systems constraints such as network size, the server typically only communicates with a subset of the clients at a time[1]. For example, in the popular `FedAvg` algorithm [51], at each communication round the server broadcasts its current model to a subset of available clients (referred to as a *cohort*), who use the model to initialize local optimization and send their model updates back to the server.

Intuitively, larger cohort sizes have the potential to improve the convergence of FL algorithms. By sampling more clients per round, we can observe a more representative sample of the underlying population—possibly reducing the number of communication rounds needed to achieve a given accuracy. This intuition is reflected in many convergence analyses of FL methods [29, 31, 32, 60, 67], which generally show that asymptotic convergence rates improve as the cohort size increases.

Larger cohorts can also provide privacy benefits. For example, when using the distributed differential privacy model [6, 11, 16, 63] in federated learning, noise is typically added to the updates sent from the clients to the server [53]. This helps preserve privacy but can also mar the utility of the learned model. By dividing the noise among more clients, larger cohorts may mitigate detrimental effects of noise. Moreover, since privacy tends to decrease as a function of the number of communication rounds [1, 17], larger cohorts also have the potential to improve privacy in FL by reducing the number of rounds needed for convergence.

Motivated by the potential benefits of large-cohort training, we systematically explore the impact of cohort size in realistic cross-device settings. Our results show that increasing the cohort size may

[1] In contrast, *cross-silo* settings often have a small set of clients, most of which participate in each round [27].

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

not lead to significant convergence improvements in practice, despite their theoretical benefit [67]. Moreover, large-cohort training can introduce fundamental optimization and generalization issues. Our results are reminiscent of work on large-batch training in centralized settings, where larger batches can stagnate convergence improvements [14, 19, 50, 68, 69], and even lead to generalization issues with deep neural networks [23, 30, 45, 46, 47, 49, 62]. While some of the challenges we identify with large-cohort training are parallel to issues that arise in large-batch centralized learning, others are unique to federated learning and have not been previously identified in the literature.

Contributions. In this work, we provide a novel examination of cohort sizes in federated learning. We give a wide ranging empirical analysis spanning many popular federated algorithms and datasets (Section 2). Despite the many possible benefits of large-cohort training, we find that challenges exist in realizing these benefits (Section 3). We show that these issues are caused in part by distinctive characteristics of federated training dynamics (Section 4). Using these insights, we provide partial solutions to the challenges we identify (Section 5), focusing on how to adapt techniques from large-batch training, and the limitations of such approaches. Our solutions are designed to serve as simple benchmarks for future work. We conclude by discussing limitations and open problems (Section 6).

1.1 Related Work

Large-batch training. In non-federated settings, mini-batch stochastic gradient descent (SGD) and its variants are common choices for training machine learning models, particularly deep neural networks. While larger mini-batch sizes ostensibly allow for improved convergence (in terms of the number of steps required to reach a desired accuracy), in practice speedups may quickly saturate when increasing the mini-batch size. This property of diminishing returns has been explored both empirically [14, 19, 50, 62] and theoretically [47, 68]. Beyond the issue of speedup saturation, numerous works have also observed a *generalization gap* when training deep neural networks with large batches [23, 30, 45, 46, 49, 69]. Our work differs from these areas by specifically exploring how the cohort size (the number of selected clients) affects *federated* optimization methods. While some of the issues with large-batch training appear in large-cohort training, we also identify a number of new challenges introduced by the federated setting.

Optimization for federated learning. Significant attention has been paid towards developing federated optimization techniques. Such work has focused on various aspects, including communication-efficiency [5, 34, 37, 51], data and systems heterogeneity [25, 28, 29, 39, 40, 41, 42, 65], and fairness [26, 43]. We provide a description of some relevant methods in Section 2, and defer readers to recent surveys such as [27] and [41] for additional background. One area pertinent to our work is that of variance reduction for federated learning, which can mitigate negative effects of data heterogeneity [28, 29, 71]. However, such methods often require clients to maintain state across rounds [29, 71], which may be infeasible in cross-device settings [27]. Moreover, such methods may not perform well in settings with limited client participation [60]. Many convergence analyses of federated optimization methods show that larger cohort sizes can lead to improved convergence rates, even without explicit variance reduction [31, 32, 67]. These analyses typically focus on asymptotic convergence, and require assumptions on learning rates and heterogeneity that may not hold in practice [9, 27]. In this work, we attempt to see whether increasing the cohort size leads to improved convergence in practical, communication-limited settings.

Client sampling. A number of works have explored how to select cohorts of a fixed size in cross-device FL [10, 12, 18, 57, 61]. Such methods can yield faster convergence than random sampling by carefully selecting the clients that participate at each round, based on quantities such as the client loss. However, such approaches typically require the server to be able to choose which clients participate in a cohort. In practice, cohort selection in cross-device federated learning is often governed by client availability, and is not controlled by the server [7, 59]. In this work we instead focus on the impact of size of the cohort, assuming the cohort is sampled at random.

2 Preliminaries

Federated optimization methods often aim to minimize a weighted average of client loss functions:

$$\min_x f(x) := \sum_{k=1}^{K} p_k f_k(x), \tag{1}$$

where K is total number of clients, the p_k are client weights satisfying $p_k \geq 0$, and f_k is the loss function of client k. For practical reasons, p_k is often set to the number of examples in client k's local dataset [42, 51].

To solve (1), each client in a sampled cohort could send $\nabla f_k(x)$ to the server, and the server could then apply (mini-batch) SGD. This approach is referred to as `FedSGD` [51]. This requires communication for every model update, which may not be desirable in communication-limited settings. To address this, McMahan et al. [51] propose `FedAvg`, in which clients perform multiple epochs of local training, potentially reducing the number of communication rounds needed for convergence.

We focus on a more general framework, `FedOpt`, introduced by Reddi et al. [60] that uses both client and server optimization. At each round, the server sends its model x to a cohort of clients C of size M. Each client $c_k \in C$ performs E epochs of training using mini-batch SGD with client learning rate η_c, producing a local model x_k. Each client $k \in C$ then communicates their *client update* Δ_k to the server, where $\Delta_k := x_k - x$ is the difference between the client's local model and the server model. The server computes a weighted average Δ of the client updates, and updates its own model via

$$x' = \textsc{ServerOpt}(x, \eta_s, \Delta)\,, \tag{2}$$

where $\textsc{ServerOpt}(x, \eta_s, g)$ is some first-order optimizer, η_s is the server learning rate, and g is a gradient estimate. For example, if $\textsc{ServerOpt}$ is SGD, then $\textsc{ServerOpt}(x, \eta_s, g) = x - \eta_s g$. The Δ in (2) is referred to as a **pseudo-gradient** [60]. While Δ may not be an unbiased estimate of ∇f, it can serve a similar role. Full pseudo-code of `FedOpt` is given in Algorithm 1.

Algorithm 1 `FedOpt` framework

Input: M, T E, x^1, η_c, η_s, $\textsc{ServerOpt}$, $\{p_k\}_{k=1}^K$
for $t = 1, \cdots, T$ **do**
  The server selects a cohort C_t of M clients uniformly at random, without replacement.
  The server sends x^t to all clients in C_t.
  Each client $k \in C_t$ performs E epochs of mini-batch SGD on f_k with step-size η_c.
  After training, each $k \in C_t$ has a local model x_k^t and sends $\Delta_k^t = x^t - x_k^t$ to the server.
  The server computes a pseudo-gradient Δ^t and updates its model via

$$\Delta^t = \frac{\sum_{k \in C_t} p_k \Delta_k^t}{\sum_{k \in C_t} p_k}, \quad x^{t+1} = \textsc{ServerOpt}(x_t, \eta_s, \Delta^t).$$

Algorithm 1 generalizes a number of federated learning algorithms, including `FedAvg` [51], `FedAvgM` [25], `FedAdagrad` [60], and `FedAdam` [60]. These are the cases where $\textsc{ServerOpt}$ is SGD, SGD with momentum, Adagrad [15, 52], and Adam [33], respectively. `FedSGD` is realized when $\textsc{ServerOpt}$ is SGD, $\eta_c = 1$, $E = 1$, and each client performs full-batch gradient descent.

2.1 Experimental Setup

We aim to understand how the cohort size M impacts the performance of Algorithm 1. In order to study this, we perform a wide-ranging empirical evaluation using various special cases of Algorithm 1 across multiple datasets, models, and tasks. We discuss the key facets of our experiments below.

Datasets, models, and tasks. We use four datasets: CIFAR-100 [35], EMNIST [13], Shakespeare [8], and Stack Overflow [3]. For CIFAR-100, we use the client partitioning proposed in [60]. The other three datasets have natural client partitions that we use. For EMNIST, the handwritten characters are partitioned by their author. For Shakespeare, speaking lines in Shakespeare plays are partitioned by their speaker. For Stack Overflow, posts on the forum are partitioned by their author. The number of clients and examples in the training and test sets are given in Table 1.

For CIFAR-100, we train a ResNet-18, replacing batch normalization layers with group normalization (as proposed and empirically validated in federated settings by Hsieh et al. [24]). For EMNIST, we train a convolutional network with two convolutional layers, max-pooling, dropout, and two dense layers. For Shakespeare, we train an RNN with two LSTM layers to perform next-character-prediction. For Stack Overflow, we perform next-word-prediction using an RNN with a single LSTM layer. For full details on the models and datasets, see Appendix A.1.

Table 1: Dataset statistics.

DATASET	TRAIN CLIENTS	TRAIN EXAMPLES	TEST CLIENTS	TEST EXAMPLES
CIFAR-100	500	50,000	100	10,000
EMNIST	3,400	671,585	3,400	77,483
SHAKESPEARE	715	16,068	715	2,356
STACK OVERFLOW	342,477	135,818,730	204,088	16,586,035

Algorithms. We implement many special cases of Algorithm 1, including `FedSGD`, `FedAvg`, `FedAvgM`, `FedAdagrad`, and `FedAdam`. We also develop two novel methods: `FedLARS` and `FedLamb`, which are the special cases of Algorithm 1 where SERVEROPT is LARS [69] and Lamb [70], respectively. See Section 5 for the motivation and full details of these algorithms.

Implementation and tuning. Unless otherwise specified, in Algorithm 1 clients perform $E = 1$ epochs of training with mini-batch SGD. Their batch size is fixed per-task. We set p_k to be the number of examples in client k's dataset. We tune learning rates for all algorithms and models using a held-out validation set: We perform $T = 1500$ rounds of training with $M = 50, E = 1$ for each algorithm and model, varying η_c, η_s over $\{10^i \mid -3 \leq i \leq 1\}$ and select the values that maximize the average validation performance over 5 random trials. All other hyperparameters (such as momentum) are fixed. For more details, see Appendix A. We provide open-source implementations of all simulations in TensorFlow Federated [4]. All experiments were conducted using clusters of multi-core CPUs, though our results are independent of wall-clock time and amount of compute resources.

Presentation of results. We apply the algorithms above to the tasks listed above with varying cohort sizes. For brevity, we present only a fraction of our results, selecting representative experiments to illustrate large-cohort training phenomena. The full set of experimental results can be found in Appendix B. We run 5 random trials for each experiment, varying the model initialization and which clients are sampled per round. In all subsequent figures, dark lines indicate the mean across the 5 trials, and shaded regions indicate one standard deviation above and below the mean.

3 Large-Cohort Training Challenges

In this section we explore challenges that exist when using large cohorts in federated learning. While some of these challenges mirror issues in large-batch training, others are unique to federated settings.

3.1 Catastrophic Training Failures

We first discuss a practical issue unique to large-cohort training. Due to data heterogeneity, the server model x, may be misaligned with some client's loss f_k, in which case $\nabla f_k(x)$ can blow up and lead to optimization problems. This issue is exacerbated by large cohorts, as we are more likely to sample misaligned clients. To demonstrate this, we applied `FedAvg` to EMNIST with varying cohort sizes M, using learning rates tuned for $M = 10$. For each M, we performed 5 random trials and recorded whether a *catastrophic training failure* occurred, in which the training accuracy decreased by a factor of at least $1/2$ in a single round.

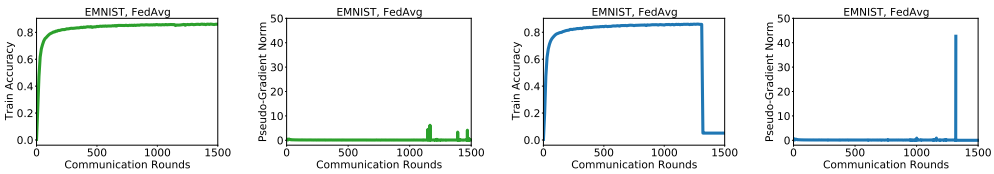


Figure 1: Applying `FedAvg` to EMNIST with cohort size 200. We plot the train accuracy and norm of the pseudo-gradient for a trial that ran successfully (**left**), and one that experienced a catastrophic training failure (**right**). The trials differed only in which clients were randomly sampled each round.

The failure rate increased from 0% for $M = 10$ to 80% for $M = 800$. When failures occured, we consistently saw a spike in the norm of the pseudo-gradient Δ (see Figure 1). In order to prevent this

spike, we apply clipping to the client updates. We use the *adaptive clipping* method of [2]. While this technique was originally designed for training with differential privacy [2], we found that it greatly improved the stability of large-cohort training. Applying `FedAvg` to EMNIST with adaptive clipping, no catastrophic training failures occurred for any cohort size. We use adaptive clipping in all subsequent experiments. For more details, see Appendix A.3.

3.2 Diminishing Returns

In this section, we show that increasing M in Algorithm 1 can lead to improved convergence, but that such improvements diminish with M. To demonstrate this, we plot the test accuracy of `FedAvg` and `FedSGD` across multiple tasks, for varying cohort sizes M. Results for CIFAR-100 and Stack Overflow are given in Figure 2, though we observe similar trends for all tasks (see Appendix B.1).

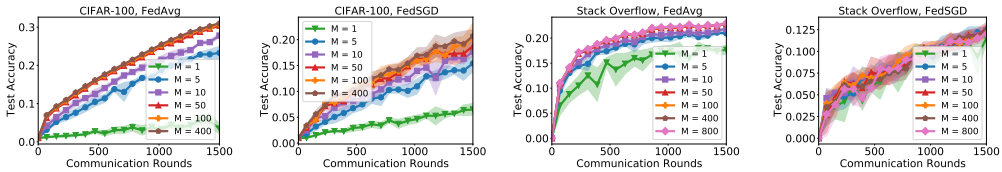


Figure 2: Test accuracy of `FedAvg`, `FedSGD` for various cohort sizes M on CIFAR-100 (left) and Stack Overflow (right).

We see that convergence benefits do not scale linearly with cohort size. While increasing M from 1 to 10 can significantly improve convergence, there is generally a threshold after which point increasing M incurs little to no change in convergence. This threshold is typically between $M = 10$ and $M = 50$. Interestingly, this seems to be true for both tasks, even though $M = 50$ represents 10% of the training clients for CIFAR-10, but only approximately 0.015% of the training clients for Stack Overflow. We see comparable results for EMNIST and Shakespeare, as well as for other optimizers, including `FedAdam` and `FedAdagrad`. See Appendix B.1 for the full results. In short, we see that increasing M alone can lead to *diminishing returns*, or even no returns in terms of convergence. This mirrors issues of diminishing returns in large-batch training [14, 19, 50, 62].

3.3 Generalization Failures

Large-batch centralized optimization methods have repeatedly been shown to converge to models with worse generalization ability than models found by small-batch methods [23, 30, 45, 46, 49, 69]. Given the parallels between batch size in centralized learning and cohort size in FL, this raises obvious questions about whether similar issues occur in FL. In order to test this, we applied `FedAvg`, `FedAdam`, and `FedAdagrad` with different cohort sizes to various models. In Figure 3, we plot the train and test accuracy of each model after $T = 1500$ communication rounds.

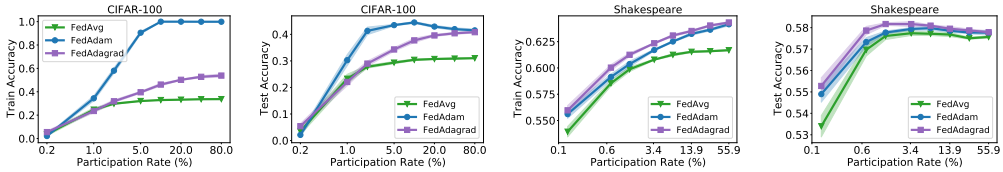


Figure 3: The train and test accuracy of `FedAvg`, `FedAdam`, and `FedAdagrad` on CIFAR-100 (left) and Shakespeare (right) after training for 1500 communication rounds, for varying cohort sizes. The x-axis denotes the percentage of training clients in each cohort.

We find that generalization issues do occur in FL. For example, consider `FedAdam` on the CIFAR-100 task. While it attains roughly the same training accuracy for $M \in \{50, 100, 200, 400\}$, we see that the larger cohorts uniformly lead to worse generalization. This resembles the findings of Keskar et al. [30], who show that generalization issues of large-batch training can occur even though the methods reach similar training losses. However, generalization issues do not occur uniformly. It is often optimizer-dependent (as in CIFAR-100) and does not occur on the EMNIST and Stack

Overflow datasets (see Appendix B.2). Notably, CIFAR-100 and Shakespeare have many fewer clients overall. Thus, large-cohort training may reduce generalization, especially when the cohort size is large compared to the total number of clients.

3.4 Fairness Concerns

One critical issue in FL is fairness across clients, as minimizing (1) may disadvantage some clients [43, 55]. Intuitively, large-cohort training methods may be better suited for ensuring fairness, since a greater fraction of the population is allowed to contribute to the model at each round. As a coarse measure of fairness, we compute percentiles of accuracy of our trained models across test clients. Fairer algorithms should lead to higher accuracy values for smaller percentiles. The percentiles for `FedAdam` on each task are given in Figure 4.

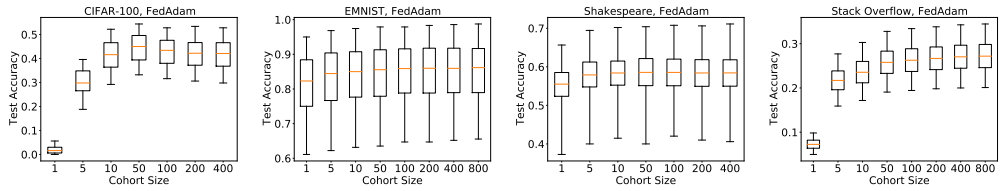


Figure 4: Accuracy of `FedAdam` after training for 1500 communication rounds using varying cohort sizes and tasks. The box plots show the 5th, 25th, 50th, 75th, and 95th percentiles of accuracy across test clients.

We find that the cohort size seems to affect all percentiles in the same manner. For example, on CIFAR-100, $M = 50$ performs better for smaller percentiles and larger percentiles than larger M. This mirrors the CIFAR-100 generalization failures from Section 3.3. By contrast, for Stack Overflow we see increases in all percentiles as we increase M. While the accuracy gains are only slight, they are consistent across percentiles. This suggests a connection between the fairness of a federated training algorithm and the fraction of test clients participating at every round. Notably, increasing M seems to have little effect on the spread between percentiles (such as the difference between the 75th and 25th percentiles) beyond a certain point. See Appendix B.3 for more results.

3.5 Decreased Data Efficiency

Despite issues such as diminishing returns and generalization failures, federated optimization methods can see some benefit from larger cohorts. Large-cohort training, especially with adaptive optimizers, often leads to faster convergence to given accuracy thresholds. For example, in Figure 5, we see that the number of rounds `FedAdam` requires to reach certain accuracy thresholds generally decreases with the cohort size.

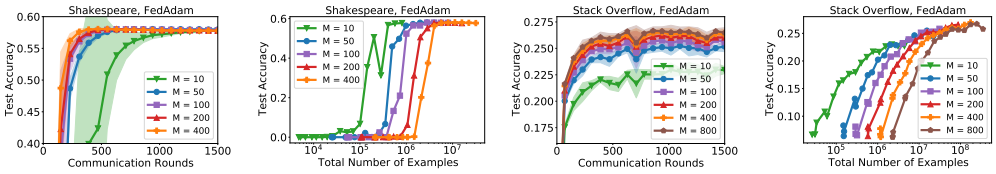


Figure 5: Test accuracy of `FedAdam` on Shakespeare (left) and Stack Overflow (right) with various cohort sizes. We plot versus the number of communication rounds and the number of examples processed in total.

While it is tempting to say that large-cohort methods are "faster", this ignores the practical costs of large-cohort training. Completing a single communication round often requires more resources with larger cohorts. To showcase this, we also plot the accuracy of `FedAdam` with respect to the number of examples seen in Figure 5. This measures the data-efficiency of large-cohort training, and shows that large cohort-training requires significantly more examples per unit-accuracy.

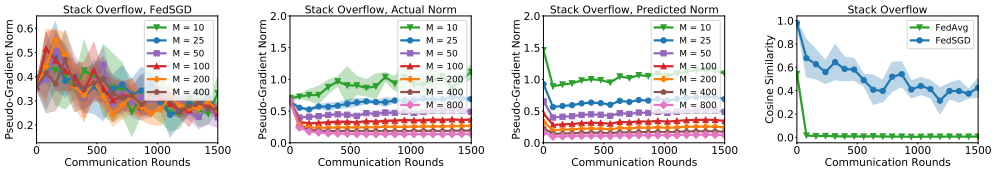
While data-ineffiency also occurs in large-batch training [50], it is especially important in federated learning. Large-cohort training faces greater limitations on parallelizability due to data-sharing

constraints. Worse, in realistic cross-device settings client compute times can scale super-linearly with their amount of data, so clients with more data are more likely to become *stragglers* [7]. This straggler effect means that data-inefficient algorithms may require longer training times. To demonstrate this, we show in Appendix B.4 that under the probabilistic straggler runtime model from [38], large-cohort training can require significantly more compute time to converge.

4 Diagnosing Large-Cohort Challenges

We now examine the challenges in Section 3, and provide partial explanations for their occurrence. One of the key differences between `FedAvg` and `FedSGD` is what the pseudo-gradient Δ in (1) represents. In `FedSGD`, Δ is a stochastic gradient estimate (*i.e.,* $\mathbb{E}[\Delta] = \nabla f$, where the expectation is over all randomness in a given communication round). For special cases of Algorithm 1 where clients perform multiple local training steps, Δ is not an unbiased estimator of ∇f [9, 48, 58]. While increasing the cohort size should reduce the variance of Δ as an estimator of $\mathbb{E}[\Delta]$, it is unclear what this quantity represents.

To better understand Δ, we plot its norm on Stack Overflow in Figures 6a and 6b. For `FedSGD`, $\|\Delta\|$ decreases slightly with M, but has high variance. By contrast, for `FedAvg` larger cohorts lead to smaller norms with little overlap. The decrease in norm obeys an inverse square root rule: Let Δ_1, Δ_2 be pseudo-gradients at some round for cohort sizes M_1, M_2. For `FedAvg`, $\|\Delta_1\|/\|\Delta_2\| \approx \sqrt{M_2/M_1}$. We use this rule to predict pseudo-gradient norms for `FedAvg` in Figure 6c. After a small number of rounds, we obtain a remarkably good approximation. To explain this, we plot the average cosine similarity between client updates Δ_k^t at each round in Figure 6d, with $M = 50$. For `FedAvg`, the client updates are almost orthogonal. This explains Figure 6b, as Δ is an average of nearly orthogonal vectors. As we show in Appendix B.5, similar results hold for other tasks and optimizers.



(a) (b) (c) (d)

Figure 6: The pseudo-gradient norm of `FedSGD` (a) and `FedAvg` (b) on Stack Overflow with varying cohort sizes M. We also plot the predicted norm for `FedAvg` using an inverse square root scaling rule relative to $M = 50$ (c) and the average cosine similarity of client updates for $M = 50$ (d).

Implications for large-cohort training. This near-orthogonality of client updates is key to understanding the challenges in Section 3. The diminishing returns in Section 3.2 occur in part because increasing M leads to smaller updates. This also sheds light on Section 3.5: In large-cohort training, we take an average of many nearly-orthogonal vectors, so each client's examples contribute little. The decreasing pseudo-gradient norms in Figure 6c also highlights an advantage of methods such as `FedAdam` and `FedAdagrad`: Adaptive server optimizers employ a form of normalization that makes them somewhat scale-invariant, compensating for this norm reduction.

5 Designing Better Methods

We now explore an initial set of approaches aimed at improving large-cohort training, drawing inspiration where possible from large-batch training.

Learning rate scaling. One common technique for large-batch training is to scale the learning rate according to the batch size. Two popular scaling methods are square root scaling [36] and linear scaling [20]. While such techniques have had clear empirical benefit in centralized training, there are many different ways that they could be adapted to federated learning. For example, in Algorithm 1, the client and server optimization both use learning rates that could be scaled.

We consider the following scaling method for large-cohort training: We fix the client learning rate, and scale the server learning rate with the cohort size. Such scaling may improve convergence by

compensating for the pseudo-gradient norm reduction in Figure 6. We use square root and linear scaling rules: Given a learning rate η_s tuned for M, for $M' \geq M$ we use a learning rate η_s' where

$$\eta_s' = \frac{\sqrt{M'}}{\sqrt{M}}\eta_s \text{ (square root scaling)} \quad \text{OR} \quad \eta_s' = \frac{M'}{M}\eta_s \text{ (linear scaling).} \tag{3}$$

We also use a version of the warmup strategy from [20]. For the first W communication rounds, we linearly increase the server learning rate from η_s to η_s'. In our experiments, we set $W = 100$ and use a reference server learning rate η_s tuned for $M = 50$. Our experiments show that server learning rate scaling rules have mixed efficacy in large-cohort training. Linear scaling is often too aggressive for federated learning, and caused catastrophic training failures beyond $M = 100$ even when using adaptive clipping (see Appendix B.6). Square root scaling did not cause catastrophic training failures. However, its performance (Figure 7) varied widely across tasks. For example, it significantly improved train accuracy on Shakespeare, but reduced test accuracy. While it led to small improvements in accuracy for some cohort sizes on Stack Overflow, it degraded accuracy for the largest cohort sizes. As we show in Appendix B.6, CIFAR-100 behaves similarly to Shakespeare, while EMNIST behaved similarly to Stack Overflow. In sum, we find that applying learning rate scaling at the server may not directly improve large-cohort training.

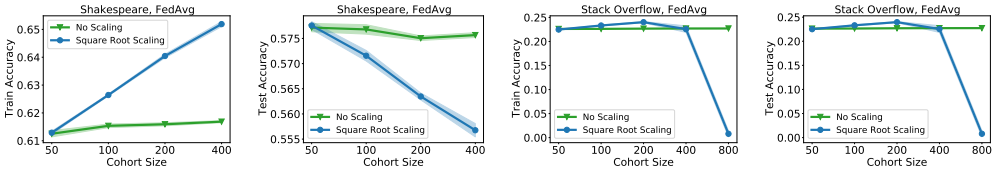


Figure 7: The train and test accuracy of `FedAvg` using square root scaling with warmup, versus no scaling. Results are given for Shakespeare (left) and Stack Overflow (right).

Layer-wise adaptivity. Another popular technique for large-batch training is *layer-wise adaptivity*. Methods such as LARS [69] and Lamb [70] use layer-wise adaptive learning rates, which may allow the methods to train faster than SGD with linear scaling and warmup in large-batch settings [69, 70]. We propose two new federated versions of these optimizers, `FedLARS` and `FedLamb`. These are special cases of Algorithm 1, where the server uses LARS and Lamb, respectively. Given the difficulties of learning rate scaling above, `FedLARS` and `FedLamb` may perform better in large-cohort settings.

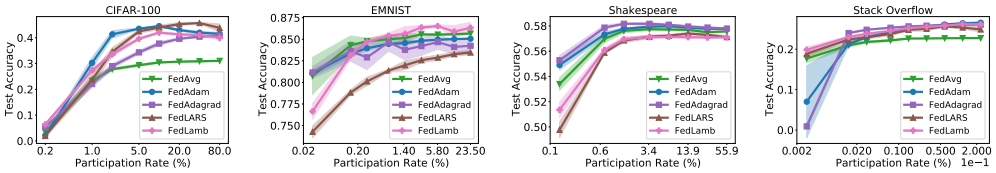


Figure 8: The test accuracy of various methods, including `FedLARS` and `FedLamb`, after training for 1500 rounds, for varying cohort sizes and on varying tasks. The x-axis denotes percentage of training clients in each cohort.

In Figure 8, we present the accuracy of various methods, including `FedLARS` and `FedLamb`, for varying cohort sizes. In most cases, we see that `FedLamb` performs comparably to `FedAdam` for large cohort sizes, but with slightly worse performance in intermediate stages. One notable exception is Stack Overflow, in which `FedLamb` performs well even for $M = 1$. As in Section 3.3, `FedLamb` sees an eventual drop in test accuracy for $M > 100$. `FedLARS` has decidedly mixed performance. While it performs well on CIFAR-10, it does not do well on EMNIST or Shakespeare. For the full results of `FedLARS` and `FedLamb`, see Appendix B.1. While federated layer-wise adaptive algorithms can be better than coordinate-wise adaptive algorithms on certain datasets in some large-cohort settings, our results do not indicate that they are universally better.

Dynamic cohort sizes. As we saw in Section 3.5, large-cohort training can reduce data efficiency. Part of this stems from the fact that larger cohorts may help very little for smaller accuracy thresholds (see Figure 5). In order to improve data efficiency, we may be able to use smaller cohorts in earlier

optimization stages, and increase the cohort size over time. This technique is parallel to "dynamic batch size" techniques used in large-batch training [64]. In order to test the efficacy of such techniques in large-cohort training, we start with an initial cohort size of $M = 50$ and double the size every 300 rounds up to $M = 800$ (or the maximum population size if smaller). This results in doubling the cohort size a maximum of 4 times over the 1500 rounds of training we perform. We plot the results for `FedAvg` and `FedAdam` on CIFAR-100 and Stack Overflow in Figure 9. See Appendix B.7 for results on all tasks.

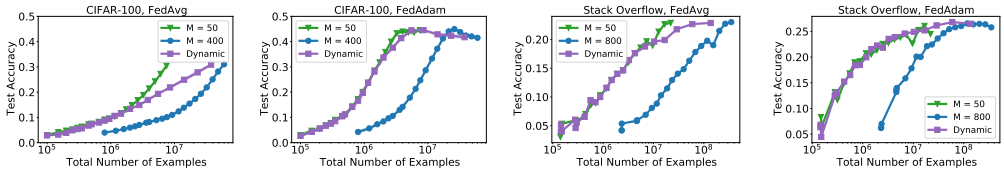


Figure 9: Test accuracy of `FedAvg` and `FedAdam` on Shakespeare (left) and Stack Overflow (right), with respect to the total number of examples processed, using fixed and dynamic cohort sizes.

This dynamic strategy attains data efficiency closer to a fixed cohort size of $M = 50$, while still obtaining a final accuracy closer to having used a large fixed cohort size. While our initial findings are promising, we note two important limitations. First, the accuracy of the dynamic strategy is bounded by the minimum and maximum cohort size used; It never attains a better accuracy than $M = 800$. Second, the doubling strategy still faces the generalization issues discussed in Section 3.3.

Hyperparameter tuning and other results. The methods discussed above, including learning rate scaling and layer-wise adaptivity, can require significant tuning to perform well [56]. To date, little work has been paid to how to tune hyperparameters in federated learning. Such work may be vital to obtain optimal performance, especially given our observations in Section 4 and the client-server structure of federated algorithms, which gives rise to many more hyperparameters. In Algorithm 1, tuning could involve the client optimizer, the client batch size, the server optimizer, and the cohort size. In fact, the client batch size is a key hyperparameter. Recall that clients perform E epochs of mini-batch SGD on their local datasets. Fixing E, the batch size dictates the number of local training steps they perform. As we show in Appendix B.8, this number of local steps is critical for achieving maximal performance, and may be necessary to tune according to the cohort size.

The near-orthogonality of client updates in Figure 6d suggests that averaging client updates may be sub-optimal. Intuitively, taking the average of random vectors on the unit sphere will result in a vector very close to zero, which may not be a useful descent direction for the server. To ameliorate this, we propose a normalized version of `FedAvg`, in which the server applies SGD to $\Delta/\|\Delta\|$. We show in Appendix B.9 that this can improve convergence in many cases, even without changing hyperparameters. This begs the question (which we leave to future work): *How should client updates be aggregated in federated learning?*

6 Limitations and Future Work

In this work we explore the benefits and limitations of large-cohort training in federated learning. As discussed in Sections 3.5 and 5, focusing on the number of communication rounds often obscures the data efficiency of a method. This in turn impacts many metrics important to society, such as total energy consumption or total carbon emissions. While we show that large-cohort training can negatively impact such metrics by reducing data-efficiency (see Section 3.5 and Appendix B.4), a more specialized focus on these issues is warranted. Similarly, we believe that an analysis of fairness in large-cohort settings going beyond Section 3.3 would be beneficial.

Future work also involves connecting large-cohort training to other important aspects of federated learning, and continuing to explore connections with growing lines of work in large-batch training. In particular, we wish to see whether noising strategies, especially differential privacy mechanisms, can help overcome the generalization issues of large-cohort training. Personalization may also help mitigate issues of generalization and fairness. Finally, although not a focus of our work, we note that some of the findings above may extend to cross-silo settings, especially if communication restrictions require subsampling clients.

References

[1] Martin Abadi, Andy Chu, Ian Goodfellow, H. Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, page 308–318, 2016.

[2] Galen Andrew, Om Thakkar, H Brendan McMahan, and Swaroop Ramaswamy. Differentially private learning with adaptive clipping. *arXiv preprint arXiv:1905.03871*, 2019.

[3] The TensorFlow Federated Authors. TensorFlow Federated Stack Overflow dataset, 2019. URL https://www.tensorflow.org/federated/api_docs/python/tff/simulation/datasets/stackoverflow/load_data.

[4] The TFF Authors. TensorFlow Federated, 2019. URL https://www.tensorflow.org/federated.

[5] Debraj Basu, Deepesh Data, Can Karakus, and Suhas Diggavi. Qsparse-local-SGD: Distributed SGD with quantization, sparsification and local computations. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

[6] Andrea Bittau, Úlfar Erlingsson, Petros Maniatis, Ilya Mironov, Ananth Raghunathan, David Lie, Mitch Rudominer, Ushasree Kode, Julien Tinnes, and Bernhard Seefeld. PROCHLO: Strong privacy for analytics in the crowd. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 441–459, 2017.

[7] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloé Kiddon, Jakub Konečný, Stefano Mazzocchi, Brendan McMahan, Timon Van Overveldt, David Petrou, Daniel Ramage, and Jason Roselander. Towards federated learning at scale: System design. In *Proceedings of Machine Learning and Systems*. Proceedings of MLSys, 2019.

[8] 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.

[9] Zachary Charles and Jakub Konečný. Convergence and accuracy trade-offs in federated learning and meta-learning. In *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*, 2021.

[10] Wenlin Chen, Samuel Horvath, and Peter Richtarik. Optimal client sampling for federated learning. *arXiv preprint arXiv:2010.13723*, 2020.

[11] Albert Cheu, Adam Smith, Jonathan Ullman, David Zeber, and Maxim Zhilyaev. Distributed differential privacy via shuffling. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 375–403, 2019.

[12] Yae Jee Cho, Jianyu Wang, and Gauri Joshi. Client selection in federated learning: Convergence analysis and power-of-choice selection strategies. *arXiv preprint arXiv:2010.01243*, 2020.

[13] 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.

[14] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Marc'aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, Quoc Le, and Andrew Ng. Large scale distributed deep networks. In *Advances in Neural Information Processing Systems*, volume 25, 2012.

[15] 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.

[16] Úlfar Erlingsson, Vitaly Feldman, Ilya Mironov, Ananth Raghunathan, Shuang Song, Kunal Talwar, and Abhradeep Thakurta. Encode, shuffle, analyze privacy revisited: Formalizations and empirical evaluation. *arXiv preprint arXiv:2001.03618*, 2020.

[17] Antonious M Girgis, Deepesh Data, Suhas Diggavi, Peter Kairouz, and Ananda Theertha Suresh. Shuffled model of federated learning: Privacy, communication and accuracy trade-offs. *arXiv preprint arXiv:2008.07180*, 2020.

[18] Jack Goetz, Kshitiz Malik, Duc Bui, Seungwhan Moon, Honglei Liu, and Anuj Kumar. Active federated learning. *arXiv preprint arXiv:1909.12641*, 2019.

[19] Noah Golmant, Nikita Vemuri, Zhewei Yao, Vladimir Feinberg, Amir Gholami, Kai Rothauge, Michael W Mahoney, and Joseph Gonzalez. On the computational inefficiency of large batch sizes for stochastic gradient descent. *arXiv preprint arXiv:1811.12941*, 2018.

[20] 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.

[21] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[22] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9 (8):1735–1780, 1997.

[23] Elad Hoffer, Itay Hubara, and Daniel Soudry. Train longer, generalize better: closing the generalization gap in large batch training of neural networks. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

[24] Kevin Hsieh, Amar Phanishayee, Onur Mutlu, and Phillip Gibbons. The non-IID data quagmire of decentralized machine learning. In *Proceedings of the 37th International Conference on Machine Learning*, 2020.

[25] 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.

[26] Zeou Hu, Kiarash Shaloudegi, Guojun Zhang, and Yaoliang Yu. FedMGDA+: Federated learning meets multi-objective optimization. *arXiv preprint arXiv:2006.11489*, 2020.

[27] Peter Kairouz, H. Brendan McMahan, and contributors. Advances and open problems in federated learning. *Foundations and Trends® in Machine Learning*, 14(1), 2021. ISSN 1935-8237.

[28] Sai Praneeth Karimireddy, Martin Jaggi, Satyen Kale, Mehryar Mohri, Sashank J Reddi, Sebastian U Stich, and Ananda Theertha Suresh. Mime: Mimicking centralized stochastic algorithms in federated learning. *arXiv preprint arXiv:2008.03606*, 2020.

[29] Sai Praneeth Karimireddy, Satyen Kale, Mehryar Mohri, Sashank Reddi, Sebastian Stich, and Ananda Theertha Suresh. SCAFFOLD: Stochastic controlled averaging for federated learning. In *Proceedings of the 37th International Conference on Machine Learning*, 2020.

[30] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. In *International Conference on Learning Representations*, 2017.

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

[32] Ahmed Khaled, Konstantin Mishchenko, and Peter Richtarik. Tighter theory for local sgd on identical and heterogeneous data. In *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, 2020.

[33] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

[34] Jakub Konečnỳ, H Brendan McMahan, Daniel Ramage, and Peter Richtárik. Federated optimization: Distributed machine learning for on-device intelligence. *arXiv preprint arXiv:1610.02527*, 2016.

[35] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

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

[37] Yassine Laguel, Krishna Pillutla, Jerôme Malick, and Zaid Harchaoui. A superquantile approach to federated learning with heterogeneous devices. In *2021 55th Annual Conference on Information Sciences and Systems (CISS)*, 2021.

[38] Kangwook Lee, Maximilian Lam, Ramtin Pedarsani, Dimitris Papailiopoulos, and Kannan Ramchandran. Speeding up distributed machine learning using codes. *IEEE Transactions on Information Theory*, 64(3):1514–1529, 2017.

[39] Daliang Li and Junpu Wang. FedMD: Heterogenous federated learning via model distillation. *arXiv preprint arXiv:1910.03581*, 2019.

[40] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smithy. FedDANE: A federated newton-type method. In *2019 53rd Asilomar Conference on Signals, Systems, and Computers*, pages 1227–1231. IEEE, 2019.

[41] Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. Federated learning: Challenges, methods, and future directions. *IEEE Signal Processing Magazine*, 37(3):50–60, 2020.

[42] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. Federated optimization in heterogeneous networks. In *Proceedings of Machine Learning and Systems 2020*, pages 429–450, 2020.

[43] Tian Li, Maziar Sanjabi, Ahmad Beirami, and Virginia Smith. Fair resource allocation in federated learning. In *International Conference on Learning Representations*, 2020.

[44] Wei Li and Andrew McCallum. Pachinko allocation: DAG-structured mixture models of topic correlations. In *Proceedings of the 23rd International Conference on Machine Learning*, 2006.

[45] Tao Lin, Sebastian U Stich, Kumar Kshitij Patel, and Martin Jaggi. Don't use large mini-batches, use local SGD. In *International Conference on Learning Representations*, 2019.

[46] Tao Lin, Lingjing Kong, Sebastian Stich, and Martin Jaggi. Extrapolation for large-batch training in deep learning. In *Proceedings of the 37th International Conference on Machine Learning*, 2020.

[47] Siyuan Ma, Raef Bassily, and Mikhail Belkin. The power of interpolation: Understanding the effectiveness of SGD in modern over-parametrized learning. In *Proceedings of the 35th International Conference on Machine Learning*, 2018.

[48] Grigory Malinovskiy, Dmitry Kovalev, Elnur Gasanov, Laurent Condat, and Peter Richtarik. From local SGD to local fixed-point methods for federated learning. In *Proceedings of the 37th International Conference on Machine Learning*, 2020.

[49] Dominic Masters and Carlo Luschi. Revisiting small batch training for deep neural networks. *arXiv preprint arXiv:1804.07612*, 2018.

[50] Sam McCandlish, Jared Kaplan, Dario Amodei, and OpenAI Dota Team. An empirical model of large-batch training. *arXiv preprint arXiv:1812.06162*, 2018.

[51] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, volume 54, 2017.

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

[53] H. Brendan McMahan, Daniel Ramage, Kunal Talwar, and Li Zhang. Learning differentially private recurrent language models. In *International Conference on Learning Representations*, 2018.

[54] Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Eleventh Annual Conference of the International Speech Communication Association*, 2010.

[55] Mehryar Mohri, Gary Sivek, and Ananda Theertha Suresh. Agnostic federated learning. In *Proceedings of the 36th International Conference on Machine Learning*, 2019.

[56] Zachary Nado, Justin M Gilmer, Christopher J Shallue, Rohan Anil, and George E Dahl. A large batch optimizer reality check: Traditional, generic optimizers suffice across batch sizes. *arXiv preprint arXiv:2102.06356*, 2021.

[57] Takayuki Nishio and Ryo Yonetani. Client selection for federated learning with heterogeneous resources in mobile edge. In *IEEE International Conference on Communications (ICC)*, 2019.

[58] Reese Pathak and Martin J Wainwright. FedSplit: An algorithmic framework for fast federated optimization. In *Advances in Neural Information Processing Systems*, volume 33, pages 7057–7066, 2020.

[59] Matthias Paulik, Matt Seigel, Henry Mason, Dominic Telaar, Joris Kluivers, Rogier van Dalen, Chi Wai Lau, Luke Carlson, Filip Granqvist, Chris Vandevelde, et al. Federated evaluation and tuning for on-device personalization: System design & applications. *arXiv preprint arXiv:2102.08503*, 2021.

[60] Sashank J. Reddi, Zachary Charles, Manzil Zaheer, Zachary Garrett, Keith Rush, Jakub Konečný, Sanjiv Kumar, and Hugh Brendan McMahan. Adaptive federated optimization. In *International Conference on Learning Representations*, 2021.

[61] Monica Ribero and Haris Vikalo. Communication-efficient federated learning via optimal client sampling. *arXiv preprint arXiv:2007.15197*, 2020.

[62] Christopher J Shallue, Jaehoon Lee, Joseph Antognini, Jascha Sohl-Dickstein, Roy Frostig, and George E Dahl. Measuring the effects of data parallelism on neural network training. *Journal of Machine Learning Research*, 20:1–49, 2019.

[63] Elaine Shi, TH Hubert Chan, Eleanor Rieffel, Richard Chow, and Dawn Song. Privacy-preserving aggregation of time-series data. In *Proc. NDSS*, volume 2, pages 1–17, 2011.

[64] Samuel L. Smith, Pieter-Jan Kindermans, and Quoc V. Le. Don't decay the learning rate, increase the batch size. In *International Conference on Learning Representations*, 2018.

[65] Hongyi Wang, Mikhail Yurochkin, Yuekai Sun, Dimitris Papailiopoulos, and Yasaman Khazaeni. Federated learning with matched averaging. In *International Conference on Learning Representations*, 2020.

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

[67] Haibo Yang, Minghong Fang, and Jia Liu. Achieving linear speedup with partial worker participation in non-IID federated learning. In *International Conference on Learning Representations*, 2021.

[68] Dong Yin, Ashwin Pananjady, Max Lam, Dimitris Papailiopoulos, Kannan Ramchandran, and Peter Bartlett. Gradient diversity: a key ingredient for scalable distributed learning. In *Proceedings of the Twenty-First International Conference on Artificial Intelligence and Statistics*, 2018.

[69] Yang You, Igor Gitman, and Boris Ginsburg. Large batch training of convolutional networks. *arXiv preprint arXiv:1708.03888*, 2017.

[70] Yang You, Jing Li, Sashank Reddi, Jonathan Hseu, Sanjiv Kumar, Srinadh Bhojanapalli, Xiaodan Song, James Demmel, Kurt Keutzer, and Cho-Jui Hsieh. Large batch optimization for deep learning: Training bert in 76 minutes. In *International Conference on Learning Representations*, 2020.

[71] Xinwei Zhang, Mingyi Hong, Sairaj Dhople, Wotao Yin, and Yang Liu. FedPD: A federated learning framework with optimal rates and adaptivity to non-IID data. *arXiv preprint arXiv:2005.11418*, 2020.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] The challenges we discuss are described in detail in Section 3, and we discuss the benefits and drawbacks of translating large-batch training techniques to large-cohort settings in Section 5.
   (b) Did you describe the limitations of your work? [Yes] These are discussed in Section 6. Moreover, our discussion in Section 5 attempts to be completely transparent about which methods worked well, and which did not, in order to give the reader a holistic sense of what works in large-cohort training.
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] These are discussed in Section 6, as well as Section 3.5.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] We include anonymized code for reproducing all experimental results in the supplement.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] We discuss the core details of our experimental setup in Section 2.1, and give all details in Appendix A.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] When applicable, all tables and plots in Sections 3, 4, and 5 give error bars with respect to 5 random trials using varying random seeds.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] As discussed in Section 2.1, we use a distributed cluster of multi-core CPUs for all experiments. However, none of our experimental results rely on any form of wall-clock time or the amount of compute time. Instead, we report things such as number of examples processed or the number of (simulated) communication rounds performed between clients and server, which do not depend on the amount of compute resources.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] All datasets and models drawn from existing assets and are given proper attributions in Sections 2.1 and A. All algorithms used that were developed prior to this work are attributed throughout.
   (b) Did you mention the license of the assets? [N/A]
   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]