Adaptive Federated Learning with Auto-Tuned Clients via Local Smoothness

Junhyung Lyle Kim [1] Mohammad Taha Toghani [2] César A. Uribe [2] Anastasios Kyrillidis [1]

Abstract

Federated learning (FL) is a distributed machine learning framework where the global model of a central server is trained via multiple collaborative steps by participating clients without sharing their data. While being a flexible framework, where the distribution of local data, participation rate, and computing power of each client can greatly vary, such flexibility gives rise to many new challenges, especially in the hyperparameter tuning on both the server and the client side. We propose Δ-SGD, a simple step size rule for SGD that enables each client to use its own step size by adapting to the local smoothness of the function each client is optimizing. We provide empirical results where the benefit of the client adaptivity is shown in various FL scenarios. In particular, our proposed method achieves TOP-1 accuracy in 73% and TOP-2 accuracy in 100% of the experiments considered without additional tuning.

1. Introduction

Federated learning (FL) is a distributed machine learning framework that allows multiple clients to learn a global model collaboratively. Each client trains the model on its local data and sends only the updated model parameters to a central server for aggregation. Mathematically, FL aims to solve the following optimization problem:

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

where $f_i(x) := \mathbb{E}_{z\sim\mathcal{D}_i}[F_i(x, z)]$ is the loss function of the i-th client, and m is the number of clients.

A key property of FL is its flexibility in *how different clients participate in the overall training*: the number of clients m, their participation rates, and computing power available to each client can all vary and change at any point during the overall training procedure. On top of that, each client's local data is not shared with other clients or the server, resulting in certain inherent data privacy (McMahan et al., 2017; Agarwal et al., 2018).

While advantageous, such flexibility and data privacy also introduces a plethora of new challenges, notably: $i)$ how the server aggregates the local information coming from each client, and $ii)$ how to make sure each client meaningfully "learns" using their local data and computing device. The first challenge was partially addressed in Reddi et al. (2021), where adaptive optimization methods, such as Adagrad (Duchi et al., 2011) or Adam (Kingma & Ba, 2014), were utilized in the aggregation step. However, the second challenge remains largely unaddressed other than the exception of Wang et al. (2021); Xie et al. (2019).

Local data of each client is not shared, which intrinsically implies heterogeneity in terms of the size and the distribution $\mathcal{D}_i$ of local datasets. That is, $\mathcal{D}_i$ differs for each client i, as well as the number of samples $z \sim \mathcal{D}_i$. Consequently, $f_i(x)$ can vastly differ from client to client, making problem (1) hard to solve. Moreover, the sheer amount of local updates is far larger than the number of aggregation steps, given that less communication is desired due to the high communication cost—typically 3-4$\times$ orders of magnitude more expensive than local computation—in distributed optimization settings (Lan et al., 2020).

Hence, extensive fine-tuning of the client optimizers is often required to achieve good performance. For instance, experimental results of the famous `FedAvg` algorithm were obtained after performing a grid-search of typically 11-13 step sizes of the clients' SGD (McMahan et al., 2017, Section 3), as SGD (and its variants) are highly sensitive to the step size (Moulines & Bach, 2011; Toulis & Airoldi, 2017; Assran & Rabbat, 2020; Kim et al., 2022). Similarly, in Reddi et al. (2021), six different client step sizes were grid-searched for different tasks, and not surprisingly, each task requires a different client step size to obtain the best result, regardless of the server-side adaptivity (Reddi et al., 2021, Table 8). Importantly, these "fine-tunings" are done under the setting that *all clients use the same step size*, which is sub-optimal, given that f_i can be different per client.

[1]Department of Computer Science, Rice University, TX, USA [2]Department of Electrical and Computer Engineering, Rice University, TX, USA. Correspondence to: Junhyung Lyle Kim <jlylekim@rice.edu>.

Proceedings of the 40th International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023. Copyright 2023 by the author(s).

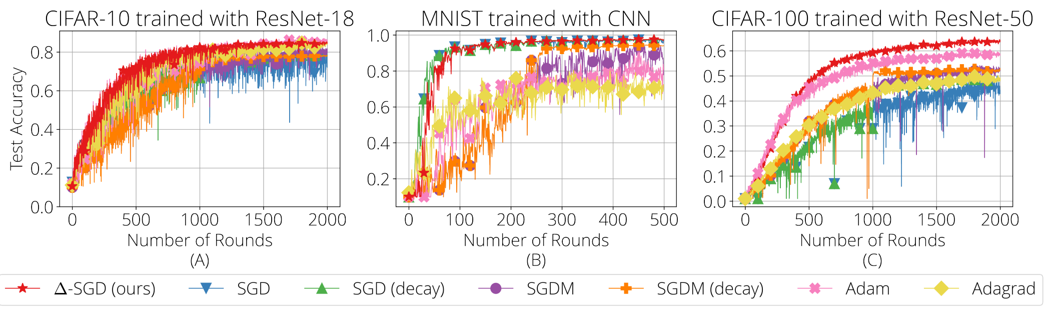

Figure 1. Test accuracies for various client optimizers for three different tasks. We fine-tune the step size for each client optimizer in task (A), and *intentionally keep it the same* for the other tasks, to highlight the effect of not properly tuning the step size of each client optimizer. Our proposed method, Δ-SGD, exhibits superior performance in all settings, without any additional tuning. (A): CIFAR-10 classification task trained on ResNet-18. (B): MNIST classification task trained on shallow CNN. (C): CIFAR-100 classification task trained on ResNet-50. All experiments use `FedAvg` (McMahan et al., 2017) as the server optimizer.

Initial examination as motivation. The implication of not properly tuning the client optimizer is highlighted in Figure 1. We plot the progress of test accuracies for different client optimizers, where, for all the other test cases, we intentionally use *the same step size rules that were fine-tuned for the task in Figure 1(A)*. There, we train a ResNet-18 for CIFAR-10 dataset classification within an FL setting, where the best step sizes were used for all algorithms after grid-search; we defer the experimental details to Section 3. Hence, all methods perform reasonably well, although Δ-SGD, our proposed method, achieves noticeably better test accuracy when compared to non-adaptive SGD variants – e.g., a 5% difference in final classification accuracy from SGDM– and comparable final accuracy only with adaptive SGD variants, like Adam and Adagrad.

In Figure 1(B), we train a shallow CNN for MNIST data classification using the same step size rules. MNIST classification is now considered an “easy” task, and hence SGD with the same constant and decaying step sizes from Figure 1(A) works well. However, with momentum, SGDM exhibits oscillating behavior, which results in slow progress and poor final accuracy, especially without decaying the step size. Adaptive optimizers like Adam and Adagrad show similar behavior: falling short in achieving good final accuracy, compared to their performance in Figure 1(A).

Similarly, in Figure 1(C), we plot the test accuracy progress for CIFAR-100 classification trained on ResNet-50, again using the same step size rules as before. Contrary to Figure 1(B), SGD with momentum (SGDM) works better than SGD, both with the constant and the decaying step sizes. Adam becomes a “good optimizer” again, but its “sibling,” Adagrad, performs worse than SGDM. On the contrary, our proposed method, Δ-SGD, which we introduce in Section 2, achieves superior performance in all cases without any additional tuning.

The above empirical observations beg answers to important and non-trivial questions in training FL tasks using variants of SGD methods as the client optimizer: *Should the momentum be used? Should the step size be decayed? If so, when?* Unfortunately, Figure 1 indicates that the answers to these questions highly vary depending on the setting; once the dataset itself or how the dataset is distributed among different clients changes, or once the model architecture changes, the client optimizers have to be properly re-tuned to ensure good performance. Perhaps surprisingly, the same holds for adaptive methods like Adagrad (Duchi et al., 2011) and Adam (Kingma & Ba, 2014).

Our hypothesis and contributions. Our paper takes a stab in this direction: we propose DELTA-SGD (**D**istribut**E**d **L**ocali**T**y **A**daptive SGD), a simple adaptive SGD scheme, that does not require any knowledge of the problem constants, such as the L-smoothness parameter. We will refer to our algorithm as Δ-SGD in the rest of the text. Our adaptive step size is extended from Malitsky & Mishchenko (2020), where the focus was centralized convex optimization; we defer the readers to Section 2 for details. Our contributions can be summarized as follows:

- We extend the (centralized) adaptive step size from Malitsky & Mishchenko (2020) to the FL setting. The implication of this adaptive step size in FL is twofold: i) each client can use its own step size, and ii) each client's step size *adapts to the local smoothness of* f_i—hence LocaliTy Adaptive— and can even *increase* during local iterations. Moreover, thanks to the simplicity of the proposed step size, our method is *agnostic to the server optimizer as well as the loss function*, and thus can be easily combined with server adaptive methods such as `FedAdam` (Reddi et al., 2021), or methods that use different (regularized) loss functions such as `FedProx` (Li et al., 2020) or `MOON` (Li et al., 2021).
- We evaluate our approach on several benchmark datasets and demonstrate that Δ-SGD achieves superior perfor-

mance compared to other state-of-the-art FL methods. Our experiments show that our method can effectively adapt the client step size to the underlying data distribution and achieve faster convergence of the global model, *without any additional tuning*. Our approach can help overcome the client step size tuning challenge in FL and enable more efficient and effective collaborative learning in distributed systems.

2. DELTA(Δ)-SGD: Distributed locality adaptive SGD

Algorithm 1 DELTA(Δ)-SGD

```
1: input: $x_0 \in \mathbb{R}^d$, $\eta_0, \theta_0, \gamma > 0$, and $p \in (0,1)$.
2: for each round $t = 0, 1, \dots, T-1$ do
3:     sample a subset $\mathcal{S}_t$ of clients with size $|\mathcal{S}_t| = p \cdot m$
4:     for each machine in parallel for $i \in \mathcal{S}_t$ do
5:         set $x_{t,0}^i = x_t$
6:         set $\eta_{t,0}^i = \eta_0$ and $\theta_{t,0}^i = \theta_0$
7:         for local step $k \in [K]$ do
8:             $x_{t,k}^i = x_{t,k-1}^i - \eta_{t,k-1}^i \tilde{\nabla} f_i(x_{t,k-1}^i)$
9:             $\eta_{t,k}^i = \min\left\{ \frac{\gamma \|x_{t,k}^i - x_{t,k-1}^i\|}{2\|\tilde{\nabla} f_i(x_{t,k}^i) - \tilde{\nabla} f_i(x_{t,k-1}^i)\|}, \sqrt{1+\theta_{t,k-1}^i}\,\eta_{t,k-1}^i \right\}$
10:            $\theta_{t,k}^i = \eta_{t,k}^i / \eta_{t,k-1}^i$
11:        end for
12:    end for
13:    $x_{t+1} = \frac{1}{|\mathcal{S}_t|} \sum_{i \in \mathcal{S}_t} x_{t,K}^i$
14: end for
15: return $x_T$
```

Malitsky & Mishchenko (2020) presented a new step size rule for (centralized) gradient descent (GD). This new method adjusts to the local geometry of the objective function. The proposed step size is clever and strikingly simple:

$$\eta_t = \min\left\{ \frac{\|x_t - x_{t-1}\|}{2\|\nabla f(x_t) - \nabla f(x_{t-1})\|}, \sqrt{1+\theta_{t-1}}\,\eta_{t-1} \right\}, \quad (2)$$

where $\theta_{t-1} = \eta_{t-1}/\eta_{t-2}$. The step size in (2) adapts to the *local smoothness* of the iterates, that is:

$$\|\nabla f(x_t) - \nabla f(x_{t-1})\| \leqslant L_t \cdot \|x_t - x_{t-1}\|, \; \forall t = 1, 2, \dots \quad (3)$$

Therefore, the step size rule in (2) can *increase during iterations*, while the second condition in (2) ensures that η_t does not increase arbitrarily. By definition, L_t in (3) is never bigger than the global smoothness parameter L, resulting in faster convergence in practice (Malitsky & Mishchenko, 2020, Section 4).

Note that (2) is vastly different from some standard adaptive methods like Adagrad (Duchi et al., 2011), which can only decrease the learning rate, as they accumulate non-negative quantities in the denominator of the step size. As a result, the scalar multiplied to their "adaptive" learning rates play a crucial role and requires proper tuning, similarly to vanilla SGD, as evident in Figure 1; see also Table 1 in Section 3.

We now introduce DELTA(Δ)-SGD: **D**istribut**E**d **L**ocali**T**y **A**daptive SGD, where we extend (2) to the FL setting, with the inclusion of stochasticity and local iterations; see Algorithm 1. For brevity, we use $\tilde{\nabla} f_i(x) = \frac{1}{|\mathcal{B}|} \sum_{z \in \mathcal{B}} F_i(x, z)$ to denote the stochastic gradients with batch size $|\mathcal{B}| = b$.

We make a few remarks about Algorithm 1. First, the input $\eta_0 > 0$ can be quite arbitrary, as it can be corrected, per client level, in the 1st local iteration (line 9); similarly for $\theta_0 > 0$ (line 10). [1] Second, as the step size $\eta_{t,k}^i$ requires two successive stochastic gradients, there could be two options: using i) the same batch or ii) a new batch; the second option will incur additional memory requirement, so we use the first.[2] Third, after local steps are done, η_0 is re-used (line 6); we could potentially use a more sophisticated approach and use a similar adaptive step size to line 9, but we leave this for future work. Last, we present simple averaging as the server-side aggregation for simplicity (line 13), but more sophisticated methods like `FedAdam` (Reddi et al., 2021) can be used instead.

3. Experimental Setup and Results

Experimental Setup. We evaluate on four datasets commonly used in FL scenarios: MNIST, FMNIST, CIFAR-10, and CIFAR-100 (Krizhevsky et al., 2009). For MNIST and FMNIST, we train a shallow CNN. For CIFAR-10, we train a ResNet-18 (He et al., 2016). For CIFAR-100, we train both a ResNet-18 and a ResNet-50 to study the effect of changing the model architecture. Due to space constraints, we defer the details of experimental setup to Appendix A.

Changing the level of non-iidness. We first investigate how the performance of different client optimizers degrade in increasing degrees of heterogeneity by varying the concentration parameter $\alpha \in \{1, 0.1, 0.01\}$ multiplied to the prior of Dirichlet distribution, following Hsu et al. (2019).

Three illustrative cases are visualized in Figure 2. We remind that the hyperparameters are tuned for the case of CIFAR-10 trained with ResNet-18, with $\alpha = 0.1$ (Figure 2(A)). For this task, with $\alpha = 1$ (i.e., closer to iid), all methods perform better than $\alpha = 0.1$, as expected. With

[1]We use the default value, $\eta_0 = 0.2$ and $\theta_0 = 1$ from the original implementation in `https://github.com/ymalitsky/adaptive_GD/blob/master/pytorch/optimizer.py` in all our experiments.

[2]This distinction was already analyzed in the centralized setting, and it was reported that using the same batch performed better in practice (Malitsky & Mishchenko, 2020).

Non-iidness	*Optimizer*	*Dataset / Model*				
$\mathrm{Dir}(\alpha \cdot \mathbf{p})$		MNIST CNN	FMNIST CNN	CIFAR-10 ResNet-18	CIFAR-100 ResNet-18	CIFAR-100 ResNet-50
$\alpha = 1$	SGD	**98.3**$_{\downarrow(0.2)}$	86.5$_{\downarrow(0.8)}$	87.7$_{\downarrow(2.1)}$	57.7$_{\downarrow(4.2)}$	53.0$_{\downarrow(12.8)}$
	SGD (↓)	97.8$_{\downarrow(0.7)}$	86.3$_{\downarrow(1.0)}$	87.8$_{\downarrow(2.0)}$	**61.9**$_{\downarrow(0.0)}$	60.9$_{\downarrow(4.9)}$
	SGDM	**98.5**$_{\downarrow(0.0)}$	85.2$_{\downarrow(2.1)}$	88.7$_{\downarrow(1.1)}$	58.8$_{\downarrow(3.1)}$	60.5$_{\downarrow(5.3)}$
	SGDM (↓)	**98.4**$_{\downarrow(0.1)}$	**87.2**$_{\downarrow(0.1)}$	**89.3**$_{\downarrow(0.5)}$	**61.4**$_{\downarrow(0.5)}$	63.3$_{\downarrow(2.5)}$
	Adam	94.7$_{\downarrow(3.8)}$	71.8$_{\downarrow(15.5)}$	**89.4**$_{\downarrow(0.4)}$	55.6$_{\downarrow(6.3)}$	61.4$_{\downarrow(4.4)}$
	Adagrad	64.3$_{\downarrow(34.2)}$	45.5$_{\downarrow(41.8)}$	86.6$_{\downarrow(3.2)}$	53.5$_{\downarrow(8.4)}$	51.9$_{\downarrow(13.9)}$
	SPS	10.1$_{\downarrow(88.4)}$	85.9$_{\downarrow(1.4)}$	82.7$_{\downarrow(7.1)}$	1.0$_{\downarrow(60.9)}$	50.0$_{\downarrow(15.8)}$
	Δ-SGD	**98.4**$_{\downarrow(0.1)}$	**87.3**$_{\downarrow(0.0)}$	**89.8**$_{\downarrow(0.0)}$	**61.5**$_{\downarrow(0.4)}$	**65.8**$_{\downarrow(0.0)}$
$\alpha = 0.1$	SGD	**98.1**$_{\downarrow(0.0)}$	83.6$_{\downarrow(2.8)}$	72.1$_{\downarrow(12.9)}$	54.4$_{\downarrow(6.7)}$	44.2$_{\downarrow(19.9)}$
	SGD (↓)	**98.0**$_{\downarrow(0.1)}$	84.7$_{\downarrow(1.7)}$	78.4$_{\downarrow(6.6)}$	59.3$_{\downarrow(1.8)}$	48.7$_{\downarrow(15.4)}$
	SGDM	**97.6**$_{\downarrow(0.5)}$	83.6$_{\downarrow(2.8)}$	79.6$_{\downarrow(5.4)}$	58.8$_{\downarrow(2.3)}$	52.3$_{\downarrow(11.8)}$
	SGDM (↓)	**98.0**$_{\downarrow(0.1)}$	**86.1**$_{\downarrow(0.3)}$	77.9$_{\downarrow(7.1)}$	60.4$_{\downarrow(0.7)}$	52.8$_{\downarrow(11.3)}$
	Adam	96.4$_{\downarrow(1.7)}$	80.4$_{\downarrow(6.0)}$	**85.0**$_{\downarrow(0.0)}$	55.4$_{\downarrow(5.7)}$	58.2$_{\downarrow(5.9)}$
	Adagrad	89.9$_{\downarrow(8.2)}$	46.3$_{\downarrow(40.1)}$	84.1$_{\downarrow(0.9)}$	49.6$_{\downarrow(11.5)}$	48.0$_{\downarrow(16.1)}$
	SPS	96.0$_{\downarrow(2.1)}$	85.0$_{\downarrow(1.4)}$	70.3$_{\downarrow(14.7)}$	42.2$_{\downarrow(18.9)}$	42.2$_{\downarrow(21.9)}$
	Δ-SGD	**98.1**$_{\downarrow(0.0)}$	**86.4**$_{\downarrow(0.0)}$	**84.5**$_{\downarrow(0.5)}$	**61.1**$_{\downarrow(0.0)}$	**64.1**$_{\downarrow(0.0)}$
$\alpha = 0.01$	SGD	96.8$_{\downarrow(0.7)}$	79.0$_{\downarrow(1.2)}$	22.6$_{\downarrow(11.3)}$	30.5$_{\downarrow(1.3)}$	24.3$_{\downarrow(7.1)}$
	SGD (↓)	**97.2**$_{\downarrow(0.3)}$	79.3$_{\downarrow(0.9)}$	**33.9**$_{\downarrow(0.0)}$	30.3$_{\downarrow(1.5)}$	24.6$_{\downarrow(6.8)}$
	SGDM	77.9$_{\downarrow(19.6)}$	75.7$_{\downarrow(4.5)}$	28.4$_{\downarrow(5.5)}$	24.8$_{\downarrow(7.0)}$	22.0$_{\downarrow(9.4)}$
	SGDM (↓)	94.0$_{\downarrow(3.5)}$	79.5$_{\downarrow(0.7)}$	29.0$_{\downarrow(4.9)}$	20.9$_{\downarrow(10.9)}$	14.7$_{\downarrow(16.7)}$
	Adam	80.8$_{\downarrow(16.7)}$	60.6$_{\downarrow(19.6)}$	22.1$_{\downarrow(11.8)}$	18.2$_{\downarrow(13.6)}$	22.6$_{\downarrow(8.8)}$
	Adagrad	72.4$_{\downarrow(25.1)}$	45.9$_{\downarrow(34.3)}$	12.5$_{\downarrow(21.4)}$	25.8$_{\downarrow(6.0)}$	22.2$_{\downarrow(9.2)}$
	SPS	69.7$_{\downarrow(27.8)}$	44.0$_{\downarrow(36.2)}$	21.5$_{\downarrow(12.4)}$	22.0$_{\downarrow(9.8)}$	17.4$_{\downarrow(14.0)}$
	Δ-SGD	**97.5**$_{\downarrow(0.0)}$	**80.2**$_{\downarrow(0.0)}$	31.6$_{\downarrow(2.3)}$	**31.8**$_{\downarrow(0.0)}$	**31.4**$_{\downarrow(0.0)}$

Table 1. Experimental results based on the settings detailed in Section 3. Performance difference within 0.5% of the best result for each task are shown in **bold**. Subscripts$_{\downarrow(x.x)}$ is the performance difference from the best result and is highlighted in pink when the difference is bigger than 2%. The symbol (↓) appended to SGD and SGDM indicates step-wise learning rate decay, where the step sizes are divided by 10 after 50%, and another by 10 after 75% of the total rounds.

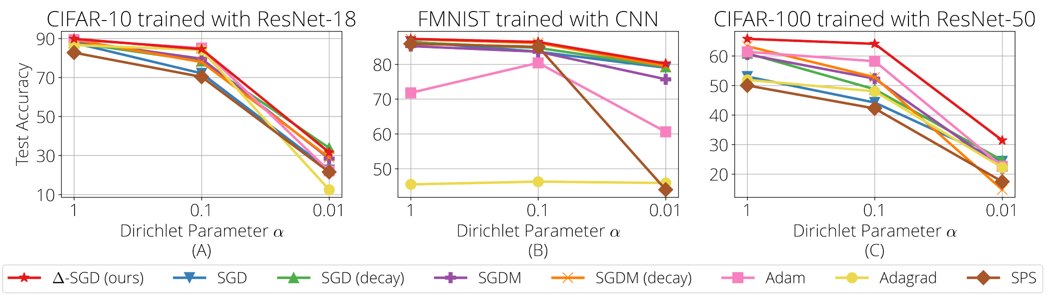

Figure 2. The effect of stronger heterogeneity on different client optimizers, induced by the Dirichlet concentration parameter α. (A): CIFAR-10 trained with ResNet-18, (B): FMNIST trained with CNN, (C): CIFAR-100 trained with ResNet-50.

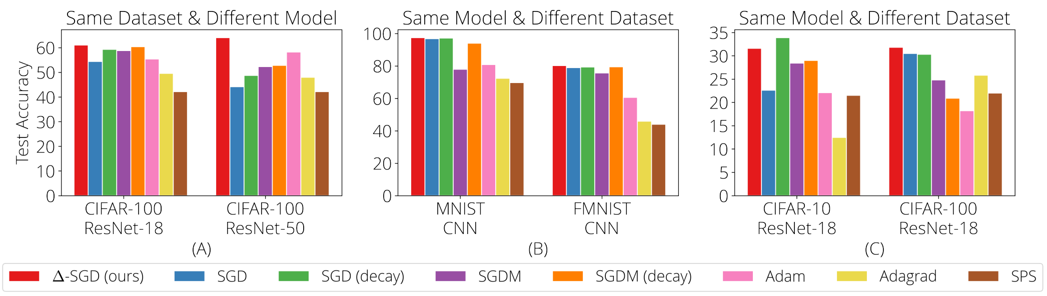

Figure 3. The effect of changing the dataset and the model architecture on different client optimizers. (A): CIFAR-100 trained with ResNet-18 versus Resnet-50 ($\alpha = 0.1$), (B): MNIST versus FMNIST trained with CNN ($\alpha = 0.01$), (C): CIFAR-10 versus CIFAR-100 trained with ResNet-18 ($\alpha = 0.01$).

$\alpha = 0.01$, which is highly non-iid, we see a significant drop in performance for all methods. SGD with LR decay and Δ-SGD perform the best, while adaptive methods like Adam ($85\% \to 22.1\%$) and Adagrad ($84.1\% \to 12.5\%$) degrade noticeably more than other methods.

For FMNIST trained with CNN (Figure 2(B)), which can be considered as an easier problem compared to CIFAR-10 classification, the performance degradation with varying α's is much milder. Interestingly, adaptive methods like Adam, Adagrad, and SPS perform much worse than other methods, indicating their inflexibility when applied to diverse datasets and model architectures, which often is worse than simple SGD-based methods; we further investigate this phenomenon in more detail in the next remark.

A similar trend can be seen in MNIST classification, where SGD-based methods work relatively well, except for SGDM with $\alpha = 0.01$: without the LR decay, SGDM only achieves around 78% accuracy, while SGD without LR decay still achieves over 96% accuracy (c.f., Figure 1(A)). Even with the LR decay, SGDM achieves noticeably worse performance. While this may seem like an "easy fix" in hindsight, it indicates that in a highly non-iid setting, one should be extremely careful in tuning SGD-based methods, even for easy datasets like MNIST.

Interestingly, SPS performs particularly poorly, merely achieving 10% for MNIST with $\alpha = 1$, while achieving 96% for the "harder" case of $\alpha = 0.1$, and again dropping to 69.7% with $\alpha = 0.01$. A possible explanation is, with simple CNN, the model is not over-parameterized enough to have $f_i^\star = 0$, the default estimate we use on which the performance of SPS crucially relies (Hazan & Kakade, 2019). Adagrad similarly performs poorly.

Lastly, in Figure 2(C), results for CIFAR-100 classification trained with ResNet-50 are illustrated. Δ-SGD exhibits superior performance in all cases of α. Unlike MNIST and FMNIST, Adam enjoys the second ($\alpha = 0.1$) or the third ($\alpha = 1$) best performance in this task, complicating how one should tune Adam for the task at hand. Other methods, including SGD with and without momentum/LR decay, Adagrad, and SPS perform much worse than Δ-SGD.

Changing the model architecture. For this remark, let us focus on CIFAR-100 trained on ResNet-18 versus on ResNet-50, with $\alpha = 0.1$, illustrated in Figure 3(A). SGD and SGDM (both with and without LR decay), Adagrad, and SPS perform worse using ResNet-50 than ResNet-18.

This is a counter-intuitive behavior, as one would expect to get better accuracy by using a more powerful model. Δ-SGD is an exception: without any additional tuning, *Δ-SGD can improve its performance.* Adam also improves similarly, but the achieved accuracy is significantly worse than that of Δ-SGD. While it is true that the performance of Δ-SGD also slightly degrades for $\alpha = 0.01$, it still performs better than the other methods.

Changing the dataset. We now focus on cases where the dataset changes, but the model architecture remains the same. We mainly consider two cases, illustrated in Figure 3(B) and (C): when CNN is trained for classifying MNIST versus FMNIST, and when ResNet-18 is trained for CIFAR-10 versus CIFAR-100. Aggravating the complexity of tuning SGD(M) for MNIST, one can observe a similar trend in CIFAR-10 trained with ResNet-18. In that case, Δ-SGD does not achieve the best test accuracy (although it is the second best with a pretty big margin with the rest), while SGD with LR decay does. However, without LR decay, the accuracy achieved by SGD drops *more than* 11%.

Interestingly, when we change the dataset from CIFAR-10 to CIFAR-100, SGD without LR decay (which is not one of the best methods for CIFAR-10) achieves the second-best performance (after Δ-SGD). On the other hand, SGDM with LR decay, which achieves the 3rd best performance for CIFAR-10, achieves very poor performance (more than 10% less than the best) for CIFAR-100. Finally, SGD with decay, which performs the best in the CIFAR-10 case, turns out to be the 3rd best algorithm when we change the dataset to CIFAR-100. Again, adaptive methods like Adam, Adagrad, and SPS perform quite poorly.

4. Conclusion

In this work, we proposed Δ-SGD, a distributed SGD scheme equipped with an adaptive step size that enables each client to use its step size and adapts to the local smoothness of the function each client is optimizing. We presented extensive empirical results, where the superiority of Δ-SGD is shown in various scenarios without any tuning. For future works, extending Δ-SGD to a coordinate-wise step size in spirit of (Duchi et al., 2011; Kingma & Ba, 2014), as well as enabling asynchronous updates (Assran et al., 2020; Toghani et al., 2022; Nguyen et al., 2022) could be interesting directions.

Broader Impacts. We believe that Δ-SGD can potentially impact reducing the amount of tuning required in training FL models. By now, it is widely believed that FL systems do not "dodge the bullet" of high carbon emission in AI/ML: as noted in a recent study (Yousefpour et al., 2023), "compute on client devices, and the communication between the clients and the server are responsible for the majority of FL's overall carbon emissions (97%)." The lack of principled hyperparameter tuning only exacerbates this problem. Based on the results presented in Section 3, Δ-SGD achieves superior performance without any additional tuning, while being robust to different settings.

References

Agarwal, N., Suresh, A. T., Yu, F. X. X., Kumar, S., and McMahan, B. cpSGD: Communication-efficient and differentially-private distributed SGD. *Advances in Neural Information Processing Systems*, 31, 2018.

Assran, M. and Rabbat, M. On the convergence of nesterov's accelerated gradient method in stochastic settings. *arXiv preprint arXiv:2002.12414*, 2020.

Assran, M., Aytekin, A., Feyzmahdavian, H. R., Johansson, M., and Rabbat, M. G. Advances in asynchronous parallel and distributed optimization. *Proceedings of the IEEE*, 108(11):2013–2031, 2020.

Duchi, J., Hazan, E., and Singer, Y. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(7), 2011.

Hazan, E. and Kakade, S. Revisiting the polyak step size. *arXiv preprint arXiv:1905.00313*, 2019.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Hsu, T.-M. H., Qi, H., and Brown, M. Measuring the effects of non-identical data distribution for federated visual classification. *arXiv preprint arXiv:1909.06335*, 2019.

Kim, J. L., Toulis, P., and Kyrillidis, A. Convergence and stability of the stochastic proximal point algorithm with momentum. In *Learning for Dynamics and Control Conference*, pp. 1034–1047. PMLR, 2022.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Krizhevsky, A., Hinton, G., et al. Learning multiple layers of features from tiny images. 2009.

Lan, G., Lee, S., and Zhou, Y. Communication-efficient algorithms for decentralized and stochastic optimization. *Mathematical Programming*, 180(1-2):237–284, 2020.

Li, Q., He, B., and Song, D. Model-contrastive federated learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10713–10722, 2021.

Li, T., Sahu, A. K., Zaheer, M., Sanjabi, M., Talwalkar, A., and Smith, V. Federated optimization in heterogeneous networks. *Proceedings of Machine learning and systems*, 2:429–450, 2020.

Loizou, N., Vaswani, S., Laradji, I. H., and Lacoste-Julien, S. Stochastic polyak step-size for sgd: An adaptive learning rate for fast convergence. In *International Conference on Artificial Intelligence and Statistics*, pp. 1306–1314. PMLR, 2021.

Malitsky, Y. and Mishchenko, K. Adaptive gradient descent without descent. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 6702–6712. PMLR, 2020.

McMahan, B., Moore, E., Ramage, D., Hampson, S., and y Arcas, B. A. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pp. 1273–1282. PMLR, 2017.

Moulines, E. and Bach, F. Non-asymptotic analysis of stochastic approximation algorithms for machine learning. *Advances in neural information processing systems*, 24, 2011.

Nguyen, J., Malik, K., Zhan, H., Yousefpour, A., Rabbat, M., Malek, M., and Huba, D. Federated learning with buffered asynchronous aggregation. In *International Conference on Artificial Intelligence and Statistics*, pp. 3581–3607. PMLR, 2022.

Paszke, A., Gross, S., Massa, e. a. F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pp. 8024–8035. Curran Associates, Inc., 2019.

Reddi, S. J., Charles, Z., Zaheer, M., Garrett, Z., Rush, K., Konečný, J., Kumar, S., and McMahan, H. B. Adaptive federated optimization. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=LkFG3lB13U5`.

Toghani, M. T., Lee, S., and Uribe, C. A. PersA-FL: Personalized Asynchronous Federated Learning. *arXiv preprint arXiv:2210.01176*, 2022.

Toulis, P. and Airoldi, E. M. Asymptotic and finite-sample properties of estimators based on stochastic gradients. 2017.

Wang, J., Xu, Z., Garrett, Z., Charles, Z., Liu, L., and Joshi, G. Local adaptivity in federated learning: Convergence and consistency. *arXiv preprint arXiv:2106.02305*, 2021.

Xie, C., Koyejo, O., Gupta, I., and Lin, H. Local adaalter: Communication-efficient stochastic gradient descent with adaptive learning rates. *arXiv preprint arXiv:1911.09030*, 2019.

Yousefpour, A., Guo, S., Shenoy, A., Ghosh, S., Stock, P., Maeng, K., Krüger, S.-W., Rabbat, M., Wu, C.-J., and Mironov, I. Green federated learning. *arXiv preprint arXiv:2303.14604*, 2023.

A. Experimental Setup

Datasets and models. To test the performance of different client optimizers in various settings, we evaluate on four datasets commonly used in FL scenarios: MNIST, FMNIST, CIFAR-10, and CIFAR-100 (Krizhevsky et al., 2009). For MNIST and FMNIST, we train a shallow CNN with two convolutional and two fully-connected layers, followed by dropout and ReLU activations. For CIFAR-10, we train a ResNet-18 (He et al., 2016). For CIFAR-100, we train both a ResNet-18 and a ResNet-50 to study the effect of changing the model architecture.

We create a federated version for each dataset by randomly partitioning the training data among 100 clients, with each client getting 500 examples.[3] To control the level of "non-iidness", we apply latent Dirichlet allocation (LDA) over the labels following Hsu et al. (2019), where each client has an associated multinomial distribution over the labels from which its examples are drawn. That is, the local training examples of each client have been drawn from a categorical distribution over N classes, parameterized by $\mathbf{q}$, such that $\mathbf{q} \sim \text{Dir}(\alpha \mathbf{p})$, where $\mathbf{p}$ is a prior distribution over N classes, with $\alpha > 0$ being the concentration parameter. We vary $\alpha \in \{0.01, 0.1, 1\}$, where bigger α indicates settings closer to i.i.d. scenarios.

FL setup and optimizers. For all cases, we fix the number of clients to be 100, and randomly sample 10% among the 100 clients. Similarly to Reddi et al. (2021); Li et al. (2020), we perform E local epochs of training over each client's dataset, and we utilize mini-batch gradients of size $b = 64$, leading to $K \approx \lfloor \frac{E \cdot 500}{64} \rfloor$ local gradient steps; we use $E = 1$ for all settings. For client optimizers, we compare stochastic gradient descent (SGD), SGD with momentum (SGDM), adaptive methods including Adam (Kingma & Ba, 2014), Adagrad (Duchi et al., 2011), SGD with stochastic Polyak step size (SPS) (Loizou et al., 2021), and our proposed method: Δ-SGD in Algorithm 1. As our focus is on client adaptivity, we only present the results using simple `FedAvg` (McMahan et al., 2017) as the server optimizer.

Hyperparameters. For all methods, we perform a simple grid search on a *single task*: CIFAR-10 classification trained with ResNet-18, with Dirichlet concentration parameter $\alpha = 0.1$; for the rest of the settings, we use the same hyperparameters. For SGD, we perform a grid search with $\eta \in \{0.01, 0.05, 0.1, 0.5\}$. For SGDM, we use the same grid for η and use momentum parameter $\beta = 0.9$. For Adam, we tried $\eta \in \{0.001, 0.01\}$, and for Adagrad, we tried $\eta \in \{0.01, 0.1\}$ (i.e., the default learning rates in Pytorch and their one-tenths). For SPS, we use the default setting of the official implementation.[4] For Δ-SGD, we append δ in front of the second condition: $\sqrt{1 + \delta\theta^i_{t,k-1}}\eta^i_{t,k-1}$ following Malitsky & Mishchenko (2020), and use $\delta = 0.1$ for all experiments. To account for the SGD(M) fine-tuning done in practice, we also tested dividing the step size (LR decay) by 10 after 50%, and then again by 10 after 75% of the total training rounds and report the inclusive results in Table 1. Finally, for the number of rounds T, we use 500 for MNIST, 1000 for FMNIST, and 2000 for CIFAR-10 and CIFAR-100.

Implementation. We use Pytorch (Paszke et al., 2019) to implement all experiments and run on eight P100 GPUs.

[3]All the datasets we consider have $50,000$ training samples, leading to 500 samples distributed per client.

[4]I.e., we use $f_i^\star = 0$, an $c = 0.5$, for the SPS step size: $\frac{f_i(x) - f_i^\star}{c\|\nabla f_i(x)\|^2}$. The official implementation can be found in `https://github.com/IssamLaradji/sps`