Towards Realistic Byzantine-Robust Federated Learning

Amit Portnoy [1] **Danny Hendler** [1]

Abstract

Federated Learning (FL) is a distributed machine learning paradigm where data is decentralized among clients who collaboratively train a model in a computation process coordinated by a central server. By assigning a weight to each client based on the proportion of data instances it possesses, the rate of convergence to an accurate joint model can be greatly accelerated. Some previous works studied FL in a Byzantine setting, where a fraction of the clients may send the server arbitrary or even malicious information regarding their model. However, these works either ignore the issue of data unbalancedness altogether or assume that client weights are known to the server a priori, whereas, in practice, it is likely that weights will be reported to the server by the clients themselves and therefore cannot be relied upon. We address this issue for the first time by proposing a practical weight-truncation-based preprocessing method and demonstrating empirically that it is able to strike a good balance between model quality and Byzantine-robustness. We also establish analytically that our method can be applied to a randomly-selected sample of client weights.

1. Introduction

Federated Learning (FL) (Konečnỳ et al., 2015; McMahan et al., 2017; Kairouz et al., 2019; Bonawitz et al., 2019) is a distributed machine learning paradigm where training data resides at autonomous client machines and the learning process is facilitated by a central server. The server maintains a shared model and alternates between requesting clients to try and improve it and integrating their suggested improvements back into that shared model.

A few interesting challenges arise from this model. First, the need for communication efficiency, both in terms of the size of data transferred and the number of required messages for reaching convergence. Second, clients are outside of the control of the server and as such may be unreliable, or even malicious. Third, while classical learning models generally assume that data is homogeneous, here privacy and the aforementioned communication concerns force us to deal with the data as it is seen by the clients; that is 1) *non-IID* (identically and independently distributed) – data may depend on the client it resides at, and 2) *unbalanced* – different clients may possess different amounts of data.

In previous works (Ghosh et al., 2019; Alistarh et al., 2018; Li et al., 2019a; Haddadpour & Mahdavi, 2019; Pillutla et al., 2019), unbalancedness is either ignored or is represented by a collection of a priori known *client importance weights* that are usually derived from the amount of data each client has. This work investigates aspects that stem from this unbalancedness. Concretely, we focus on the case where unreliable clients declare the amount of data they have and may thus adversely influence their importance weight. We show that without some mitigation, a single malicious client can obstruct convergence in this manner even in the presence of popular FL defense mechanisms. Our experiments consider protections that replace the server step by a robust mean estimator, such as median (Chen et al., 2017; Yin et al., 2018; Chen et al., 2019a) and trimmed mean (Yin et al., 2018).

The rest of this paper is organized as follows. In Section 2, we present required definitions and formalize the problem addressed by this work. Section 3 presents our truncation-based preprocessing method and proves that it can be applied to a randomly-selected sample of client weights . In Section 4, we report on the results of our empirical evaluation. Conclusions and directions for future work are presented in Section 5.

2. Problem Setup

2.1. Optimization Goal

We are given K clients where each client k has a local collection Z_k of n_k samples taken IID from some unknown distribution over sample space $\boldsymbol{Z}$. Our objective is *global* empirical risk minimization (ERM) for some loss function

[1]Ben-Gurion University. Correspondence to: Amit Portnoy <amit.portnoy@gmail.com>, Danny Hendler <hendlerd@cs.bgu.ac.il>.

class $\ell(w;\cdot)\colon \boldsymbol{Z} \to \mathbb{R}$, parameterized by $w \in \mathbb{R}^d$ [1]:

$$\min_{w\in\mathbb{R}^d} F(w), \tag{1}$$

where

$$Z = \bigcup_{k\in[K]} Z_k;\ n = |Z|;\ F(w) \coloneqq \frac{1}{n}\sum_{z\in Z}\ell(w;z).$$

2.2. Collaboration Model

We restrict ourselves to the FL paradigm, which leaves the training data distributed on client machines, and learns a shared model by iterating between client updates and server aggregation.

Additionally, an α-proportion of clients can be Byzantine, meaning they can send arbitrary and possibly malicious results on their local updates. Moreover, unlike previous works, we also consider clients' sample sizes to be unreliable because they are reported by possibly Byzantine clients. Values that are sent by clients are marked with an overdot, signifying that they are unreliable (e.g., $\dot{n}_k$).

2.3. Federated Learning Meta Algorithm

We build upon the baseline federated averaging algorithm ($FedAvg$) described by McMahan et al. (2017). There, it is suggested that in order to save communication rounds clients perform multiple stochastic gradient descent (SGD) steps while a central server occasionally averages the parameter vectors.

The intuition behind this approach becomes clearer when we mark the k^{th} client's ERM objective function by $F_k(w) \coloneqq \frac{1}{n_k}\sum_{z\in Z_k}\ell(w;z)$ and observe that the objective function in equation (1) can be rewritten as a weighted average of clients' objectives:

$$F(w) \coloneqq \frac{1}{n}\sum_{k\in[K]} n_k F_k(w).$$

Similarly to previous works (Pillutla et al., 2019; Chen et al., 2019b;a), we capture a large set of algorithms by abstracting $FedAvg$ into a meta-algorithm for FL (Algorithm 2). We require three procedures to be specified by any concrete algorithm:

(a) $Preprocess$ – receives, possibly byzantine, $\dot{n}_k$'s from clients and produces secure estimates marked as $\tilde{n}_k$'s. To the best of our knowledge, previous works ignore this procedure and assume n_k's are correct.

(b) $ClientUpdate$ – per-client w_k computation. In $FedAvg$, this corresponds to a few local mini-batch SGD rounds. See Algorithm 1 for pseudocode.

(c) $Aggregate$ – the server's strategy for updating w. In $FedAvg$, this corresponds to the weighted arithmetic mean, i.e., $w \leftarrow \frac{1}{\dot{n}}\sum_{k\in[K]}\dot{n}_k\dot{w}_k$.

Algorithm 1 $FedAvg$: $ClientUpdate$

Hyperparameters: learning rate (η), number of epochs (E), and batch size (B).

for E epochs **do**
  for B-sized batch $\mathcal{B} \subseteq Z_k$ **do**
    $w_k \leftarrow w_k - \eta\frac{1}{B}\sum_{z\in\mathcal{B}}\nabla\ell(w_k;z)$
  end for
end for

Algorithm 2 Federated Learning Meta-Algorithm

Given procedures: $Preprocess$, $ClientUpdate$, and $Aggregate$.

$\{\dot{n}_k\}_{k\in[K]} \leftarrow$ collect sample size from each client
$\{\tilde{n}_k\}_{k\in[K]} \leftarrow Preprocess(\{\dot{n}_k\}_{k\in[K]})$
$w \leftarrow$ initial guess
for $t \leftarrow 1$ **to** T **do**
  $S_t \leftarrow$ a random set of client indices
  for all $k \in S_t$ **do**
    $\dot{w}_k \leftarrow ClientUpdate(\tilde{n}_k, w)$
  end for
  $w \leftarrow Aggregate(\{\langle\tilde{n}_k, \dot{w}_k\rangle\}_{k\in S_t})$
end for

3. Preprocessing Client-Declared Sample Sizes

3.1. Preliminaries

We use $\boldsymbol{N}$ to denote the vector containing each client's declared sample size, $\boldsymbol{N} = (\dot{n}_1, \dot{n}_2, \ldots, \dot{n}_K)$. For a given upper bound on the proportion of Byzantine clients, denoted by α, we define $top(\boldsymbol{N}, \alpha)$ as the collection of the αK largest values in $\boldsymbol{N}$. We assume, w.l.o.g., that $\boldsymbol{N}$ is sorted in increasing order and that $\alpha K \in \mathbb{N}$. Additionally, we define $mwp(\boldsymbol{N}, \alpha)$ – the *maximal weight proportion* an α-proportion of clients can have when considering $\boldsymbol{N}$ as a weight vector[2]:

$$\mathrm{mwp}(\boldsymbol{N}, \alpha) \coloneqq \frac{\sum \mathrm{top}(\boldsymbol{N}, \alpha)}{\sum \boldsymbol{N}}.$$

[1] We note that some previous FL works specify a more generic finite-sum objective (McMahan et al., 2017). However, this work investigates client-declared sample sizes, whose meaning is clear under the ERM interpretation but seems meaningless in the finite-sum objective setting.

[2] $\sum \boldsymbol{X}$ denotes the sum of all values in $\boldsymbol{X}$.

Previous works (Ghosh et al., 2019; Alistarh et al., 2018; Li et al., 2019a; Haddadpour & Mahdavi, 2019; Zhao et al., 2018) either assumed that all clients have the same sample size or that there exists some constant $\alpha^* \leq 1/2$ such that $\mathrm{mwp}(\boldsymbol{N}, \alpha) < \alpha^*$. Both assumptions are unrealistic, because in practice algorithms trust the value each client reports and use it in their *Aggregate* procedure, implying that even a single malicious client can force $\mathrm{mwp}(\boldsymbol{N}, \alpha)$ to be arbitrarily close to 1. For this reason, we require that any $Preprocess$ procedure ensures the following:

$$\mathrm{mwp}(Preprocess(\boldsymbol{N}), \alpha) \leq \alpha^*. \tag{2}$$

3.2. Truncating the Values of $\boldsymbol{N}$

Our suggested preprocessing procedure uses element-wise truncation of the values of $\boldsymbol{N}$ by some value U, marked $\mathrm{trunc}(\boldsymbol{N}, U) \coloneqq \{\min(n, U) : n \in \boldsymbol{N}\}$. When given α, the maximal proportion of Byzantine clients, and a *desired* maximal weight proportion α^*, we search for the appropriate maximal truncation, U:

$$\arg\max_{U \in \mathbb{N}} \; s.t. \;\; \mathrm{mwp}(\mathrm{trunc}(\boldsymbol{N}, U), \alpha) \leq \alpha^*. \tag{3}$$

Generally, the desired α^* is an analytical property of any specific $ClientUpdate$ and $Aggregate$ combination while α and U present a trade-off. Higher α means more Byzantine tolerance but requires smaller truncation value U, which, as we demonstrate empirically, may cause slower and less accurate convergence.

We note that given α and α^*, truncating $\boldsymbol{N}$ by solving (3) is optimal in the sense that any other $Preprocess$ procedure that adheres to (2) has an equal or larger $\boldsymbol{L}^1$ distance from the original $\boldsymbol{N}$. This follows easily from the observation that, when truncating the values of $\boldsymbol{N}$, the entire distance is due to the truncated elements and if there was another applicable vector closer to $\boldsymbol{N}$, we could have increased U in contradiction to its maximally.

3.2.1. Finding a Maximal U Given α

If one has an estimate for α it is easy to calculate U. For example, by going over values in $\boldsymbol{N}$ in a decreasing order (i.e., from index K downwards) until finding a value that satisfies the inequality in (3). Then we mark the index of this value by u and use the fact that in the range $[n_u, n_{u+1}]$ we can express $\mathrm{mwp}(\mathrm{trunc}(\boldsymbol{N}, U), \alpha)$ as a simple function of the form $\frac{a+bU}{c+dU}$:

$$\frac{\sum_{i \leq u \wedge n_i \in \mathrm{top}(\boldsymbol{N}, \alpha)} n_i + |\{n_i : i > u \wedge n_i \in \mathrm{top}(\boldsymbol{N}, \alpha)\}| U}{\sum_{i \leq u} n_i + |\{n_i : i > u\}| U},$$

Algorithm 3 Report (α, U) Pairs

```
α ← α*
for u ← 1 to K − 1 do
  while mwp(trunc(N, n_{u+1}), α) > α* do
    U ← solve (4) for U ∈ [n_u, n_{u+1}]
    report (α, U)
    α ← α − 1/K
  end while
end for
```

for which we can solve (3) with

$$U \leftarrow \left\lfloor \frac{a - c\alpha^*}{d\alpha^* - b} \right\rfloor. \tag{4}$$

3.2.2. The α-U Trade-off

When we do not know α, as a practical procedure, we suggest plotting U as a function of α. In order to do so we can start with $\alpha \leftarrow \alpha^*$, $U \leftarrow n_1$, and alternate between decreasing α by $1/K$ (one less Byzantine client tolerated) and solving (3). This procedure can be made efficient by saving intermediate sums and using a specialized data structure for trimmed collections. See Algorithm 3 for pseudocode and Figure 1 for an example output.

3.3. Truncation Given a Partial View of $\boldsymbol{N}$

When K is very large we may want to sample only $k \ll K$ elements IID from $\boldsymbol{N}$. In this case we will need to test that the inequality in (3) holds with high probability.

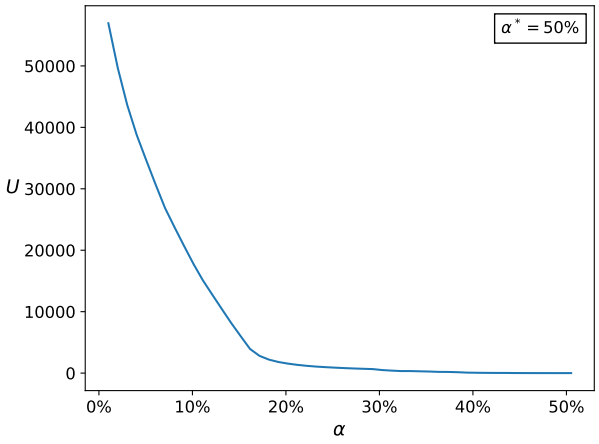

Figure 1. Example plot of data generated by executing Algorithm 3 on unbalanced vector $\boldsymbol{N}$ and $\alpha^* = 50\%$ (this vector corresponds to the partition used in our experiments; See Section 4.1 for details).

We consider k discrete random variables taken IID from $\boldsymbol{N}$ after truncation by U, that is, taken from a distribution over $\{0, 1, \ldots, U\}$. We mark these random variables as $X_1, X_2, \ldots, X_k$, and their order statistic as $X_{(1)}, X_{(2)}, \ldots, X_{(k)}$ where $X_{(1)} \leq X_{(2)} \leq \cdots \leq X_{(k)}$.

Theorem 3.1. *Given parameter $\delta > 0$ and $\varepsilon_1 = \sqrt{\frac{\ln(3/\delta)}{2k}}$, $\varepsilon_2 = U\sqrt{\frac{\ln\ln(3/\delta)}{2(k(\alpha-\varepsilon_1)+1)}}$, $\varepsilon_3 = U\sqrt{\frac{\ln\ln(3/\delta)}{2k}}$, we have that $\mathrm{mwp}(\mathrm{trunc}(\boldsymbol{N}, U), \alpha) \leq \alpha^*$ is true with $1-\delta$ confidence if the following holds:*

$$\frac{\alpha\left(\frac{\sum_{i\leftarrow\lceil(1-(\alpha-\varepsilon_1))k\rceil}^{k} X_{(i)}}{k-\lceil(1-(\alpha-\varepsilon_1))k\rceil+1} + \varepsilon_2\right)}{\left(\frac{1}{k}\sum_{i\in[k]} X_i - \varepsilon_3\right)} \leq \alpha^* .$$

Proof. First we observe that $\mathrm{mwp}(\mathrm{trunc}(\boldsymbol{N}, U), \alpha) \leq \alpha^*$ can be rewritten as

$$\begin{aligned}\mathrm{mwp}(\mathrm{trunc}(\boldsymbol{N}, U), \alpha) &= \frac{\sum \mathrm{top}(\mathrm{trunc}(\boldsymbol{N}, U), \alpha)}{\sum \mathrm{trunc}(\boldsymbol{N}, U)} \\ &= \frac{\alpha\mathbb{E}[\mathrm{top}(\mathrm{trunc}(\boldsymbol{N}, U), \alpha)]}{\mathbb{E}[\mathrm{trunc}(\boldsymbol{N}, U)]} \leq \alpha^* . \quad (5)\end{aligned}$$

Then we note that membership in $\mathrm{top}(\mathrm{trunc}(\boldsymbol{N}, U), \alpha)$ can be viewed as a simple Bernoulli random variable with probability α, for which we obtain the following bound using Hoeffding's inequality, $t \geq 0$:

$$\begin{aligned}\Pr\left[\left|\{i \in [k] : X_i \in \mathrm{top}(\mathrm{trunc}(\boldsymbol{N}, U), \alpha)\}\right| \leq (\alpha - t)k\right] \\ \leq \mathrm{e}^{-2t^2 k} .\end{aligned}$$

Therefore with $t = \varepsilon_1$, we have the following with $1 - \frac{\delta}{3}$ confidence:

$$\begin{aligned}&\sum \mathrm{top}(\{X_i \mid i \in [k]\}, \alpha) \\ &\leq \sum \{X_{(i)} \mid \lceil(1-(\alpha-\varepsilon_1))k\rceil \leq i \leq k\} . \quad (6)\end{aligned}$$

Using Hoeffding's inequality again, we can bound the expectation of $X_{(i)} \mid \lceil(1-(\alpha-\varepsilon_1))k\rceil \leq i \leq k$ by ε_2 with $1 - \frac{\delta}{3}$ confidence and together with (6) have that:

$$\begin{aligned}&\mathbb{E}[\mathrm{top}(\mathrm{trunc}(\boldsymbol{N}, U), \alpha)] \\ &\leq \frac{\sum_{i\leftarrow\lceil(1-(\alpha-\varepsilon_1))k\rceil}^{k} X_{(i)}}{k - \lceil(1-(\alpha-\varepsilon_1))k\rceil + 1} + \varepsilon_2 . \quad (7)\end{aligned}$$

Then, using Hoeffding's inequality for the third time, $\mathbb{E}[\mathrm{trunc}(\boldsymbol{N}, U)]$ is bound from below by ε_3 with $1 - \frac{\delta}{3}$ confidence:

$$\mathbb{E}[\mathrm{trunc}(\boldsymbol{N}, U)] \geq \frac{1}{k}\sum_{i\in[k]} X_i - \varepsilon_3 . \quad (8)$$

The proof is concluded by applying (6-8) to (5) using the union bound.

□

3.4. Convergence Notes

After applying our $Preprocess$ procedure we have the truncated number of samples per client, marked $\{\tilde{n}_k\}_{k\in[K]}$. We can trivially ensure that any algorithm instance works as expected by requiring that clients ignore samples that were truncated. That is, even if an honest client k (non-Byzantine) has n_k samples it may use only $\tilde{n}_k$ samples during its $ClienUpdate$.

Although this solution always preserves the semantics of *any* underlying algorithm, it does hurt convergence guarantees since the total number of samples decreases (Kairouz et al. 2019, Tables 5 and 6; Yin et al. 2018; Haddadpour & Mahdavi 2019). Interestingly, Li et al. (2019b, Theorem 3) analyze the baseline *FedAvg* and show that its convergence bound decreases with $max\, n_k$ (marked there as ν). This suggests that in some cases truncation itself may mitigate the decrease in total sample size.

Lastly, we note that in practice, the performance of federated averaging based algorithms improves when honest clients use all their original n_k samples. Intuitively, this follows easily from the observation that $Aggregate$ procedures are generally composite mean estimators and $ClientUpdate$ calls are likely to produce more accurate results given more samples.

4. Evaluation

In this section we demonstrate how truncating $\boldsymbol{N}$ is a crucial requirement for Byzantine robustness. That is, we show that no matter what is the specific attack or aggregation method, using $\boldsymbol{N}$ "as-is" categorically devoids any robustness guaranties.

4.1. Experimental Setup

A supervised learning task We train classifiers for the MNIST database (LeCun et al., 2010). It includes 28×28 grayscale labeled images of handwritten digits split into a 60,000 training set and a 10,000 test set.

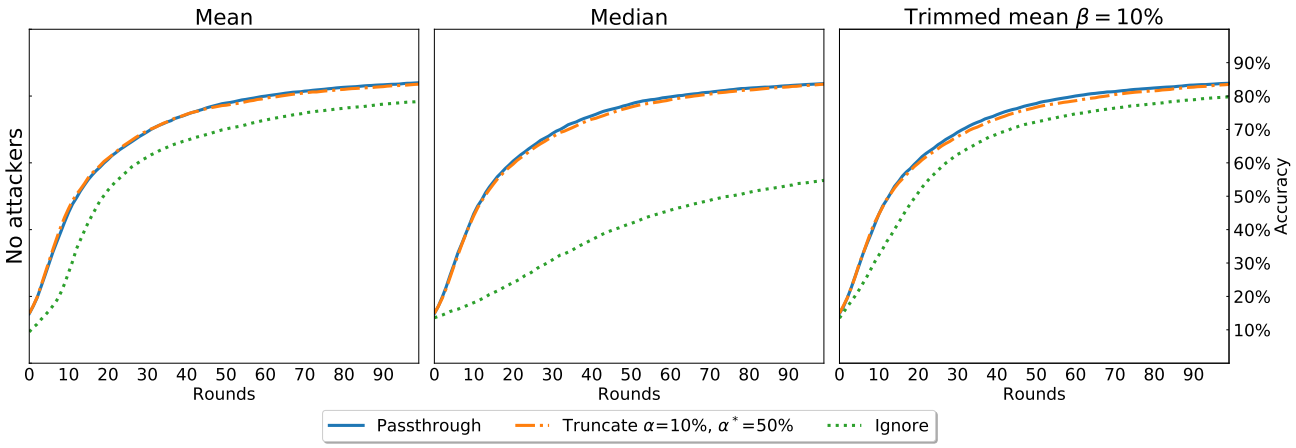

Figure 2. Accuracy by round without any attackers. Curves correspond to preprocessing procedures and columns correspond to different aggregation methods. It can be seen that our method (dashed orange curve) remains comparable to the properly weighted mean estimators (solid blue curve) while ignoring clients' sample sizes (dotted green curve) is sub-optimal. This effect is pronounced when the unweighted median is used, since with our unbalanced partition it is generally very far from the mean.

Data distribution We randomly partition the training set among 100 clients. The partition sizes are determined by taking 100 samples from a Lognormal distribution with $\mu = 1.5$, $\sigma = 3.45$, and then interpolating corresponding integers that sum to 60,000. This produces a right-skewed, fat-tailed partition size distribution that emphasizes the importance of correctly weighting aggregation rules and the effects of truncation. See Figure 3 for a histogram of the partition.

Client update Clients train using a 64-units perceptron with RelU activation and 20% dropout, followed by a softmax layer. Following Yin et al. (2018), on every communication round, all clients perform mini-batch SGD with 10% of their examples.

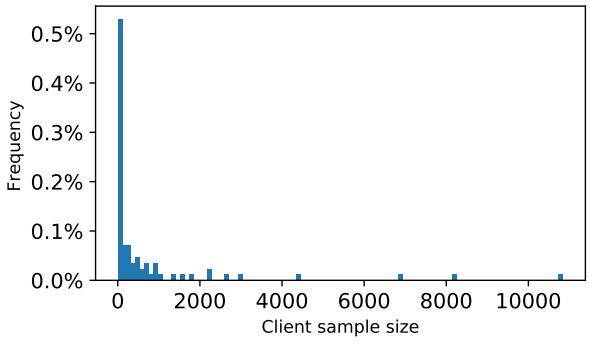

Figure 3. Histogram of the sample partition used in the experiments.

Server aggregation We show three $Aggregate$ procedures. Arithmetic mean, as used by the original $FedAvg$, and two additional procedures that replace the arithmetic mean with robust mean estimators. The first of the latter uses the coordinatewise median (Chen et al., 2017; Yin et al., 2018). That is, each server model coordinate is taken as the median of the clients' corresponding coordinates. The second robust aggregation method uses the coordinatewise trimmed mean (Yin et al., 2018) that, for a given hyperparameter β, first removes β-proportion lowest and β-proportion highest values in each coordinate and only then calculates the arithmetic mean of the remaining values.

Preprocessing client-declared sample size We either ignore client sample size, truncate according to $\alpha = 10\%$ and $\alpha^* = 50\%$, or just passthrough client sample size as reported.

Attack In order to provide comparability, we follow the experiment shown by Yin et al. (2018) in which 10% of the clients use a *label shifting attack*. In this attack, Byzantine clients train normally except for the fact that they replace every training label y with $9 - y$. The values sent by these clients are then incorrect but are relatively moderate in value making their attack somewhat harder to detect. Additionally, we examine a *model negation attack* (Blanchard et al., 2017). In this attack, each attacker "pushes" the model towards zero by always returning a negation of the server's model. When the data distribution is balanced, this attack is easily neutralized since Byzantine clients typically send extreme values. However, in our unbalanced case, we demonstrate that without our preprocessing step, this attack cannot be

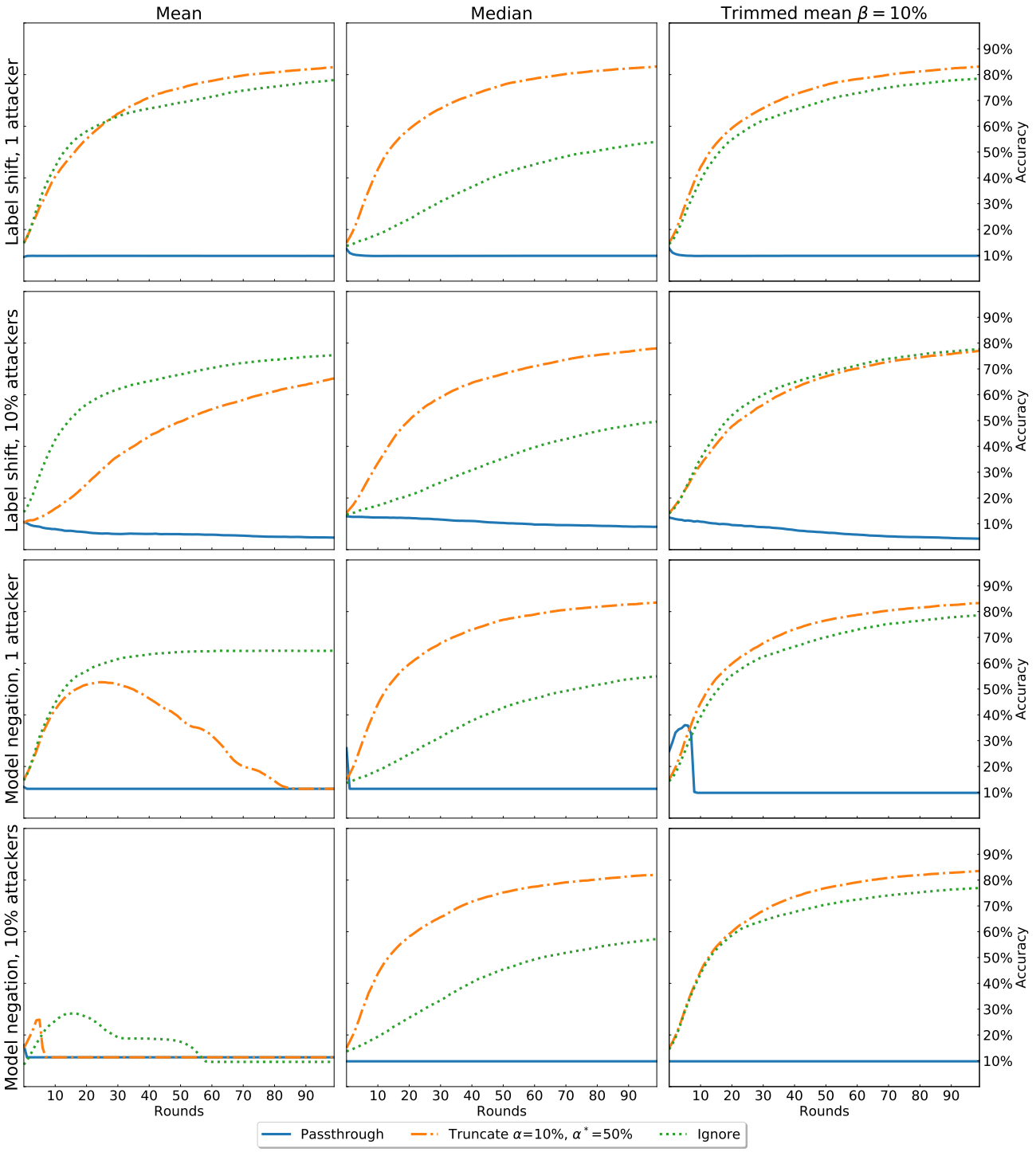

Figure 4. Accuracy by round under Byzantine attacks. Curves correspond to preprocessing procedures and columns correspond to different aggregation methods. In the first two rows Byzantine clients perform a label shifting attack with one and 10% attackers, respectively. In the last two rows we repeat the experiment with a model negation attack.

We observe that even with a single attacker performing a trivial attack (first and third rows), using the weights directly (solid blue curve) is devastating while when our preprocessing method is used in conjunction with robust mean aggregations (dashed orange curve, two last columns) convergence remain stable even when there are actual α (=10%) attackers (second and forth rows). We note that in some cases our method may be slightly less efficient compared with the preprocessing method that ignores sample size altogether (dotted green curve, second row, last column). This is to be expected because we allow Byzantine clients to potentially get close to α^*-proportion (50%, in this case) of the weight. However, our method is significantly closer to the optimal solution when there are no or only a few attackers (see Figure 2). Moreover, when used in conjunction with robust mean aggregation methods it maintains their robustness properties.

Table 1. Final test accuracies (%).

PREPROCESSING	ATTACK		AGGREGATION MEAN	MEDIAN	TRIMMED MEAN
IGNORE	NONE		78.37	54.76	79.77
	LABEL SHIFT	1	77.87	53.9	78.54
		10%	75.33	49.6	77.66
	MODEL NEGATION	1	64.8	54.9	78.56
		10%	9.58	57.12	76.94
PASSTROUGH	NONE		83.98	83.73	83.83
	LABEL SHIFT	1	9.77	9.81	9.81
		10%	4.66	8.85	4.26
	MODEL NEGATION	1	11.35	11.35	9.8
		10%	11.35	9.8	9.8
TRUNCATE	NONE		83.56	83.56	83.49
	LABEL SHIFT	1	82.85	83.15	83.14
		10%	66.31	77.95	77.0
	MODEL NEGATION	1	11.35	83.53	83.3
		10%	11.35	81.99	83.55

mitigated even by robust aggregation methods.

We first execute our experiment without any attacks for every server aggregation and preprocessing combination. Then, for each attack type, we repeat the process two additional times: 1) with a single attacker that declares 10 million samples, and 2) with 10% attackers that declare 1 million samples each.

4.2. Results

The experiment without any attackers is shown in Figure 2, the executions with attackers are shown in Figure 4 and the final test accuracies from all experiments are summarized in Table 1.

The results from the first experiment, running without any attackers (Figure 2), demonstrate that ignoring client sample size results in reduced accuracy, especially when median aggregation is used, whereas truncating according to our procedure is significantly better and is in par with properly using all weights. These results highlight the imperativeness of using sample size weights when performing server aggregations.

While Figure 2 shows that the truncation-based preprocessing performs in par with that of taking all weights into consideration when all clients are honest, Figure 4 demonstrates that the results are very different when there is an attack. In this case, we see that when even a single attacker reports a highly exaggerated sample size and the server relies on all the values of $\boldsymbol{N}$, the performance of all aggregation methods including robust median and trimmed mean quickly degrade to less than a random classifier. This is the case for both the label shifting and the model negation attacks. The label shifting attack has been shown to be mitigated by robust aggregation in the balanced setting and the model negation attack is even simpler to mitigate in the balanced setting because, as previously mentioned, Byzantine clients that employ it often send extreme values.

In contrast, in our experiments robustness is maintained when truncation-based preprocessing is used in conjunction with robust mean aggregations, even when Byzantine clients attain the maximal supported proportion ($\alpha = 10\%$).

5. Conclusion and Future Work

Our method is based on truncating the weight values reported by clients in a manner that bounds from above the proportion α^* of weights that can be attributed to Byzantine clients, given an upper bound on the proportion of clients α that may be Byzantine. Different values of parameter α represent different points in the trade-off between model quality and Byzantine-robustness, where higher values increase robustness when attacks do occur but decrease convergence rate even in the lack of attacks.

We evaluated the performance of our truncation method empirically when applied as a preprocessing stage, prior to several aggregation methods, against two types of Byzantine attacks: label shifting attack and model negation attack. The results of our experiments establish that: 1) in the absence of attacks, model convergence is in par with that of properly using all reported weights, and 2) when attacks do occur, the performance of combining truncation-based preprocessing and robust aggregations incurs almost no penalty in com-

parison with the performance of using of all weights in the lack of attacks, whereas without preprocessing, even robust aggregation methods collapse to performance worse than that of a random classifier.

When the number of clients is very large, performing server preprocessing and aggregation on the server may become computationally infeasible. We prove that, in this case, truncation-based preprocessing achieves the same upper bound on α^* w.h.p. based on the weight values reported from a sufficiently large number of the clients selected IID.

In future work, we plan to further analyze the trade-off between robustness and the usage of client sample size in rectifying data unbalancedness. We also plan to investigate alternative forms of estimating client importance that may avoid the usage of client sample size altogether.

References

Alistarh, D., Allen-Zhu, Z., and Li, J. Byzantine stochastic gradient descent. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 31*, pp. 4613–4623. Curran Associates, Inc., 2018. URL http://papers.nips.cc/paper/7712-byzantine-stochastic-gradient-descent.pdf.

Blanchard, P., El Mhamdi, E. M., Guerraoui, R., and Stainer, J. Machine learning with adversaries: Byzantine tolerant gradient descent. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 30*, pp. 119–129. Curran Associates, Inc., 2017.

Bonawitz, K., Eichner, H., Grieskamp, W., Huba, D., Ingerman, A., Ivanov, V., Kiddon, C., Konecny, J., Mazzocchi, S., McMahan, H. B., et al. Towards federated learning at scale: System design. *arXiv preprint arXiv:1902.01046*, 2019.

Chen, X., Chen, T., Sun, H., Wu, Z. S., and Hong, M. Distributed training with heterogeneous data: Bridging median- and mean-based algorithms, 2019a.

Chen, Y., Su, L., and Xu, J. Distributed statistical machine learning in adversarial settings: Byzantine gradient descent. *Proc. ACM Meas. Anal. Comput. Syst.*, 1(2), December 2017. doi: 10.1145/3154503. URL https://doi.org/10.1145/3154503.

Chen, Y., Sun, X., and Jin, Y. Communication-efficient federated deep learning with asynchronous model update and temporally weighted aggregation, 2019b.

Ghosh, A., Hong, J., Yin, D., and Ramchandran, K. Robust federated learning in a heterogeneous environment. *CoRR*, abs/1906.06629, 2019. URL http://arxiv.org/abs/1906.06629.

Haddadpour, F. and Mahdavi, M. On the convergence of local descent methods in federated learning, 2019.

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

Konečnỳ, J., McMahan, B., and Ramage, D. Federated optimization: Distributed optimization beyond the datacenter. *arXiv preprint arXiv:1511.03575*, 2015.

LeCun, Y., Cortes, C., and Burges, C. MNIST handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2, 2010.

Li, L., Xu, W., Chen, T., Giannakis, G. B., and Ling, Q. Rsa: Byzantine-robust stochastic aggregation methods for distributed learning from heterogeneous datasets. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 1544–1551, 2019a.

Li, X., Huang, K., Yang, W., Wang, S., and Zhang, Z. On the convergence of fedavg on non-iid data. *arXiv preprint arXiv:1907.02189*, 2019b.

McMahan, H. 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, 2017.

Pillutla, K., Kakade, S. M., and Harchaoui, Z. Robust aggregation for federated learning, 2019.

Yin, D., Chen, Y., Kannan, R., and Bartlett, P. Byzantine-robust distributed learning: Towards optimal statistical rates. In Dy, J. and Krause, A. (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 5650–5659, Stockholmsmssan, Stockholm Sweden, 10–15 Jul 2018. PMLR.

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