Towards Federated Learning With Byzantine-Robust Client Weighting

Amit Portnoy[1 [0000−0001−6491−5814]], Yoav Tirosh[1 [0000−0002−9469−938X]], and Danny Hendler[1 [0000−0001−7152−7828]]

Ben Gurion University of the Negev, Beer Sheva, Israel
amitport@post.bgu.ac.il

Abstract. *Federated Learning* (FL) is a distributed machine learning paradigm where data is distributed 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, in which a fraction of the clients may send 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 a priori known to the server, 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.

Keywords: Federated Learning · Machine Learning · Robustness.

1 Introduction

Federated Learning (FL) [13,17,12,4] 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 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 [9,2,15,10,18], 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 [8,21,6] and trimmed mean [21].

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 $\boldsymbol{Z}$. We denote the unified sample collection as $Z = \bigcup_{k\in[K]} Z_k$ and the total number of samples as n (i.e., $n = |Z| = \sum_{k\in[K]} n_k$). Our objective is *global* empirical risk minimization (ERM) for some loss function class $\ell(w;\cdot)\colon \boldsymbol{Z} \to \mathbb{R}$, parameterized by $w \in \mathbb{R}^d$ [1]:

$$\min_{w\in\mathbb{R}^d} F(w), \text{ where } F(w) \coloneqq \frac{1}{n}\sum_{z\in Z} \ell(w;z). \tag{1}$$

In the following sections we denote the vector of client sample sizes as $\boldsymbol{N} = (n_1, n_2, \ldots, n_K)$ and assume, w.l.o.g., that it is sorted in increasing order.

2.2 Collaboration Model

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

Additionally, a subset of the clients, marked $\mathcal{B}$, can be Byzantine, meaning they can send arbitrary and possibly malicious results on their local updates.

[1] We note that some previous FL works specify a more generic finite-sum objective [17]. 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.

Moreover, unlike previous works, we also consider clients' sample sizes to be unreliable because they are reported by possibly Byzantine clients. When the distinction is important, values that are sent by clients are marked with an overdot to signify that they are unreliable (e.g., $\dot{n}_k$), while values that have been preprocessed in some way are marked with a tilde (e.g., $\tilde{n}_k$).

2.3 Federated Learning Meta Algorithm

We build upon the baseline federated averaging algorithm (*FedAvg*) described by [17]. 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:

$$F(w) \coloneqq \frac{1}{n} \sum_{k \in [K]} n_k F_k(w). \tag{2}$$

Similarly to previous works [18,7,6], 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:

1. *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 that the n_k's are correct.
2. *ClientUpdate*—per-client w_k computation. In *FedAvg*, this corresponds to a few local mini-batch SGD rounds. See Algorithm 1 for pseudocode.
3. *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 b_k in Z_k **do**
    $w_k \leftarrow w_k - \eta \frac{1}{B} \sum_{z \in b_k} \nabla \ell(w_k; z)$
  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 clients
$\{\tilde{n}_k\}_{k\in[K]} \leftarrow Preprocess(\{\dot{n}_k\}_{k\in[K]})$
$w \leftarrow$ 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

The following assumption is common among works on Byzantine robustness:

Assumption 1 (Bounded Byzantine proportion) *The proportion of clients who are Byzantine is bounded by some constant α; i.e., $\frac{1}{K}|\mathcal{B}| \leq \alpha$.*

The next assumption is a natural generalization when considering unbalancedness:

Assumption 2 (Bounded Byzantine weight proportion) *The proportion between the combined weight of Byzantine clients and the total weight is bounded by some constant α^*; i.e., $\frac{1}{n}\sum_{b\in\mathcal{B}} n_b \leq \alpha^*$.*

Previous works on robust aggregation [9,2,15,10,22] either used Assumption 1, without considering the unbalancedness of the data, or implicitly used Assumption 2. However, we observe that Assumption 2 is unattainable in practice since Byzantine clients can often influence their weight. We address this gap with the following definition and an appropriate *Preprocess* procedure.

Definition 1 (mwp)**.** *Given a proportion p, and a weights vector $\boldsymbol{V} = (v_1, ..., v_{|\boldsymbol{V}|})$ sorted in increasing order, the* maximal weight proportion*, $\mathrm{mwp}(\boldsymbol{V}, p)$, is the maximum combined weight for* any *p-proportion of the values of $\boldsymbol{V}$:*

$$\mathrm{mwp}(\boldsymbol{V}, p) \coloneqq \frac{1}{\sum_{v\in\boldsymbol{V}} v} \sum_{(1-p)|\boldsymbol{V}|<i} v_i.$$

Note that this is just the weight proportion of the $p|\boldsymbol{V}|$ clients with the largest sample sizes.

In the rest of this work we assume Assumption 1 and design a *Preprocess* procedure that ensures the following:

$$\text{mwp}(Preprocess(\boldsymbol{N}), \alpha) \leq \alpha^*. \tag{3}$$

Observe that this requirement enables the use of weighted robust mean estimators in a realistic setting by ensuring that Assumption 2 holds for the preprocessed client sample sizes. Also note that here, α is our assumption about the proportion of Byzantine clients while α^* relates to an analytical property of the underlying robust algorithm. For example, we may replace the federated average with a weighted median as suggested by [8], in which case, α^* must be less than $1/2$.

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

$$\text{trunc}(\boldsymbol{N}, U) = (\min(n_1, U), \min(n_2, U), \ldots, \min(n_K, U)).$$

Given α and α^*, we search for the maximal truncation which satisfies (3):

$$U^* \coloneqq \arg\max_{U \in \mathbb{N}} \; s.t. \; \text{mwp}(\text{trunc}(\boldsymbol{N}, U), \alpha) \leq \alpha^*. \tag{4}$$

Here α and U^* present a trade-off. Higher α means more Byzantine tolerance but requires smaller truncation value U^*, which, may cause slower and less accurate convergence, as we demonstrate empirically in Section 4 and theoretically in Theorem 2.

We note that given α and α^*, truncating $\boldsymbol{N}$ by solving (4) is optimal in the sense that any other *Preprocess* procedure that adheres to (3) has an equal or larger $\boldsymbol{L}^1$ distance from the original $\boldsymbol{N}$. This follows immediately 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 redistributed the difference to the largest elements and increase U^* in contradiction to its maximality.

Finding 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 (4). 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 $\text{mwp}(\text{trunc}(\boldsymbol{N}, U), \alpha)$ as a simple function of the form $\frac{a+bU}{c+dU}$:

$$\frac{\sum_{(1-\alpha)K < i \leq u} n_i + |\{n_i : i > \max(u, (1-\alpha)K)\}|U}{\sum_{i \leq u} n_i + |\{n_i : i > u\}|U},$$

for which we can solve (4) with

$$U^* \leftarrow \left\lfloor \frac{a - c\alpha^*}{d\alpha^* - b} \right\rfloor. \tag{5}$$

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 (4). 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.

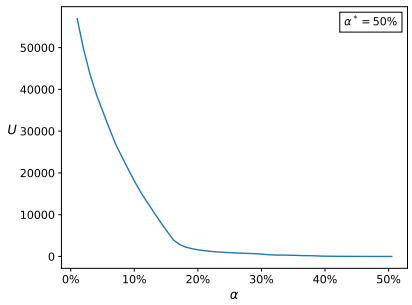

Fig. 1: Example plot of data generated by executing Algorithm 3 on unbalanced vector $\boldsymbol{N}$ and $\alpha^* = 50\%$ (this vector corresponds to the partition used in our experiments; See Section 4.1 for details).

Algorithm 3 Report (α, U^*) Pairs

$\alpha \leftarrow \alpha^*$
for $u \leftarrow 1$ **to** $K-1$ **do**
  while $\text{mwp}(\text{trunc}(\boldsymbol{N}, n_{u+1}), \alpha) > \alpha^*$ **do**
    $U^* \leftarrow$ solve (5) for $U \in [n_u, n_{u+1}]$
    report (α, U^*)
    $\alpha \leftarrow \alpha - \frac{1}{K}$
  end while
end for

3.3 Truncation given a partial view of 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 (4) holds with high probability.

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 $X_{(1)} \le X_{(2)} \le \cdots \le X_{(k)}$.

Theorem 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* $\text{mwp}(\text{trunc}(\boldsymbol{N}, U), \alpha) \le \alpha^*$ *is true with* $1-\delta$ *confidence if the following holds:*

$$\frac{\alpha\left(\frac{\sum_{i\leftarrow\lceil(1-(\alpha-\varepsilon_1))k\rceil}^{k} X_{(i)}}{k-\lceil(1-(\alpha-\varepsilon_1))k\rceil+1} + \varepsilon_2\right)}{\left(\frac{1}{k}\sum_{i\in[k]} X_i - \varepsilon_3\right)} \le \alpha^*. \tag{6}$$

Proof. First, in the scope of this proof we use a couple of additional notations:

- $\text{top}(\boldsymbol{V}, p)$: The collection of $p|\boldsymbol{V}|$ largest values in $\boldsymbol{V}$.

– $\sum \boldsymbol{V}$: The sum of all elements in $\boldsymbol{V}$.

We observe that $\text{mwp}(\text{trunc}(\boldsymbol{N}, U), \alpha) \leq \alpha^*$ can be rewritten as

$$\begin{aligned} &\text{mwp}(\text{trunc}(\boldsymbol{N}, U), \alpha) = \frac{\sum \text{top}(\text{trunc}(\boldsymbol{N}, U), \alpha)}{\sum \text{trunc}(\boldsymbol{N}, U)} = \\ &\frac{\alpha \mathbb{E}[\text{top}(\text{trunc}(\boldsymbol{N}, U), \alpha)]}{\mathbb{E}[\text{trunc}(\boldsymbol{N}, U)]} \leq \alpha^* \end{aligned} \tag{7}$$

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

$$\Pr\left[|\{i \in [k] : X_i \in \text{top}(\text{trunc}(\boldsymbol{N}, U), \alpha)\}| \leq (\alpha - t)k\right] \leq \mathrm{e}^{-2t^2 k} \tag{8}$$

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

$$\sum \text{top}(\{X_i \mid i \in [k]\}, \alpha) \leq \sum \{X_{(i)} \mid \lceil (1 - (\alpha - \varepsilon_1))k \rceil \leq i \leq k\} \tag{9}$$

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 (9) have that:

$$\mathbb{E}[\text{top}(\text{trunc}(\boldsymbol{N}, U), \alpha)] \leq \frac{\sum_{i \leftarrow \lceil (1-(\alpha-\varepsilon_1))k \rceil}^{k} X_{(i)}}{k - \lceil (1-(\alpha-\varepsilon_1))k \rceil + 1} + \varepsilon_2 \tag{10}$$

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

$$\mathbb{E}[\text{trunc}(\boldsymbol{N}, U)] \geq \frac{1}{k} \sum_{i \in [k]} X_i - \varepsilon_3 \tag{11}$$

The proof is concluded by applying (9-11) to (7) using the union bound.

3.4 Convergence Analysis

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 (non-Byzantine) client k 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 [Tables 5 and 6 in [12]; [21]; [10]]. Interestingly, Theorem 3 in [16] analyze the baseline *FedAvg* and show that the convergence bound increases with

$max\, n_k / min\, n_k$ (marked there as ν/ς). This suggests that in some cases, unbalancedness itself deteriorates the convergence rate, a phenomenon that may be mitigated by truncation to some degree.

Additionally, 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.

Lastly, as we have mentioned before, convergence is guaranteed, but we note that the optimization goal itself is inevitably skewed in our Byzantine scenario. The following theorem bounds this difference between the original weighted optimization goal (2) and the new goal after truncation. In order to emphasize the necessity of this bound (in terms of Assumption 2), we use overdot and tilde to signify unreliable and truncated values, respectively, as previously described in Subsection 2.2.

Theorem 2. *Given the same setup as in* (1) *and a truncation bound* U*, the following holds for all* $w \in \mathbb{R}^d$*:*

$$\|\frac{1}{\dot{n}} \sum_{i\in[K]} \dot{n}_i F_i(w) - \frac{1}{\tilde{n}} \sum_{i\in[K]} \tilde{n}_i F_i(w)\| \leq$$
$$\|\sum_{i:\dot{n}_i>U} \big(\frac{\dot{n}_i}{\dot{n}} - \frac{1}{K}\big) F_i(w) + \big(\frac{1}{\dot{n}} - \frac{1}{\tilde{n}}\big) \sum_{i:\dot{n}_i\leq U} \mathcal{L}(Z_i)\|$$

Where $\mathcal{L}(Z_i)$ *is defined as* $\sum_{z\in Z_i} \ell(w; z)$*.*

Proof. Using the fact that $\tilde{n} \leq UK$ we get:

$$\|\frac{1}{\dot{n}} \sum_{i\in[K]} \dot{n}_i F_i(w) - \frac{1}{\tilde{n}} \sum_{i\in[K]} \tilde{n}_i F_i(w)\| =$$
$$\|\sum_{i:\dot{n}_i>U} \frac{\dot{n}_i}{\dot{n}} F_i(w) + \frac{1}{\dot{n}} \sum_{i:\dot{n}_i\leq U} \mathcal{L}(Z_i) - \sum_{i:\dot{n}_i>U} \frac{U}{\tilde{n}} F_i(w) - \frac{1}{\tilde{n}} \sum_{i:\dot{n}_i\leq U} \mathcal{L}(Z_i)\| \leq$$
$$\|\sum_{i:\dot{n}_i>U} \frac{\dot{n}_i}{\dot{n}} F_i(w) + \frac{1}{\dot{n}} \sum_{i:\dot{n}_i\leq U} \mathcal{L}(Z_i) - \sum_{i:\dot{n}_i>U} \frac{1}{K} F_i(w) - \frac{1}{\tilde{n}} \sum_{i:\dot{n}_i\leq U} \mathcal{L}(Z_i)\| =$$
$$\|\sum_{i:\dot{n}_i>U} \big(\frac{\dot{n}_i}{\dot{n}} - \frac{1}{K}\big) F_i(w) + \big(\frac{1}{\dot{n}} - \frac{1}{\tilde{n}}\big) \sum_{i:\dot{n}_i\leq U} \mathcal{L}(Z_i)\|.$$

From the bound in Theorem 2 we can clearly see how the coefficients in the left term, $(\dot{n}_i/\dot{n} - 1/K)$, stem from unbalancedness in the values above the truncation threshold while the coefficient in the right term, $(1/\dot{n} - 1/\tilde{n})$, accounts for the increase of relative weight of the values below the truncation threshold. Additionally, note that this formulation demonstrates how a single Byzantine client can increase this difference arbitrarily by increasing its $\dot{n}_i$. Lastly, observe how both terms vanish as U increases, which motivates our selection of U^* as the *maximal* truncation threshold for any given α and α^*.

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 the specific attack or aggregation method is, using $\boldsymbol{N}$ "as-is" categorically devoids any robustness guaranties.

The code for the experiments is based on the Tensorflow machine learning library [1]. Specifically, the code for the shakespeare experiments is based on the Tensorflow Federated sub-library of Tensorflow. It is given under the Apache license 2.0. Our code can be found in the supplementary material and is given under the MIT license. We perform the experiments using a single NVIDIA GeForce RTX 2080 Ti GPU, but the results are easily reproducible on any device.

4.1 Experimental Setup

The Machine Learning Tasks and Models

Shakespeare: next-character-prediction partitioned by speaker. Presented in the original *FedAvg* paper [17] and also part of the LEAF benchmark [5], the Shakespeare dataset contains 422,615 sentences taken from *The Complete Works of William Shakespeare* [20] (freely available public domain texts). The next-character-prediction task with the per-speaker partitioning represents a realistic scenario in the FL domain. Each client trains using an LSTM recurrent model [11] with hyperparameters matching those suggested by [19] for *FedAvg*.

MNIST: digit recognition with synthetic client partitioning. The MNIST database [14] (available under Creative Commons Attribution-ShareAlike 3.0 license) includes 28×28 grayscale labeled images of handwritten digits split into 60,000 training images and 10,000 testing images. 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. Clients train a classifier using a 64-unit perceptron with RelU activation and 20% dropout, followed by a softmax layer. Following [21], on every communication round, all clients perform mini-batch SGD with 10% of their examples.

Note that the Shakespeare and MNIST synthetic tasks were selected because they are relatively simple, unbalanced tasks. Simple, because we want to evaluate a preprocessing phase and avoid tuning of the underlying algorithms we compare. Unbalanced, since as can be understood from Theorem 2, when the client sample sizes are spread mostly evenly, ignoring the client sample size altogether is a viable approach. See Figure 2 for the histograms of the partitions.

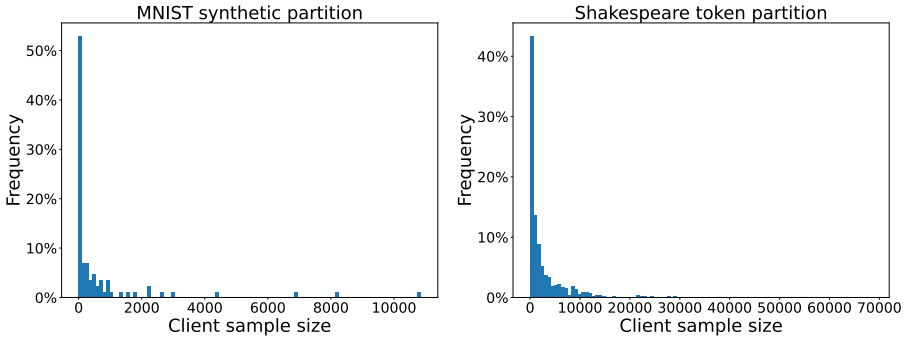

Fig. 2: Histogram of the sample partitions of the MNIST (left) and Shakespeare (right) datasets.

The Server 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 [8,21]. 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 [21] 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.

When preprocessing the client-declared sample size, we compare three options: We either ignore client sample size, truncate according to $\alpha = 10\%$ and $\alpha^* = 50\%$, or just passthrough client sample size as reported.

The Clients and Attackers We examine a *model negation attack* [3]. 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 easily detectable extreme values. However, in our unbalanced case, we demonstrate that without our preprocessing step, this attack cannot be mitigated even by robust aggregation methods.

In order to provide comparability, we additionally follow the experiment shown by [21] in which 10% of the clients use a *label shifting attack* on the MNIST task. 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.

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.

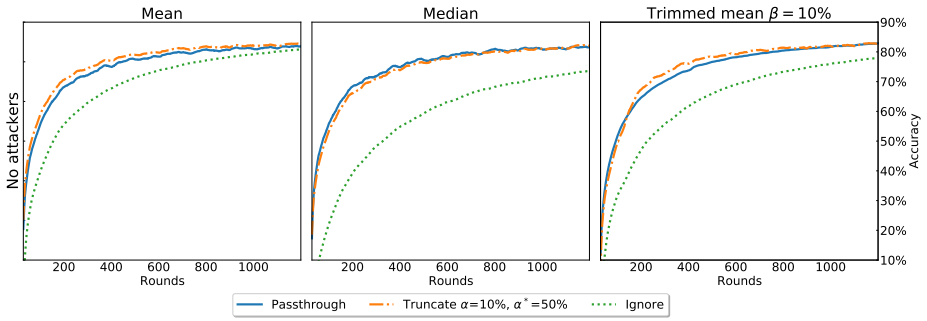

Fig. 3: Accuracy by round without any attackers for the Shakespeare experiments. Curves correspond to preprocessing procedures and columns correspond to different aggregation methods. It can be seen that our method (dashed orange curve) remains comparable to the properly weighted mean estimators (solid 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. Figure 5 shows similar results for the MNIST experiments.

4.2 Results

The Shakespeare experiments without any attackers is shown in Figure 3 and the executions with attackers are shown in Figure 4. The MNIST experiments without any attackers is shown in Figure 5 and the executions with attackers are shown in Figure 6.

The results from the first experiment, running without any attackers (Figure 3), 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 on par with properly using all weights. These results highlight the imperativeness of using sample size weights when performing server aggregations.

While Figure 3 shows that truncation-based preprocessing performs on 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 degrades.

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 ($\alpha = 10\%$).

The results of the MNIST experiments are similar to the results of the Shakespeare experiments.

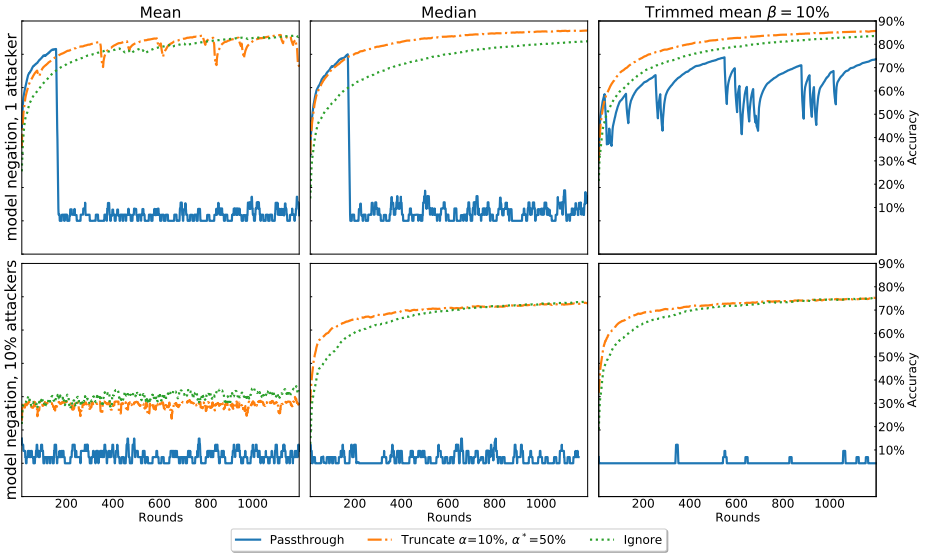

Fig. 4: Accuracy by round under Byzantine attacks for the Shakespeare experiments. Curves correspond to preprocessing procedures and columns correspond to different aggregation methods. In the two rows of the experiment the Byzantine clients perform a model negation attack with one and 10% attackers, respectively.

We observe that even with a single attacker performing a trivial attack (first row), 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 remains stable even when there are actual α (=10%) attackers (second row). In contrast, the same cannot be said for the regular mean aggregator, as can be seen by the sub-optimal accuracy (2nd row) and occasional dips in accuracy (1st row) in the leftmost column (the dips can be explained by the fact that in each round we randomly select clients for training, and so the byzantine clients have varying effects across different rounds). 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, middle 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 3). Moreover, when used in conjunction with robust mean aggregation methods it maintains their robustness properties. Figure 6 shows similar results for the MNIST experiments.

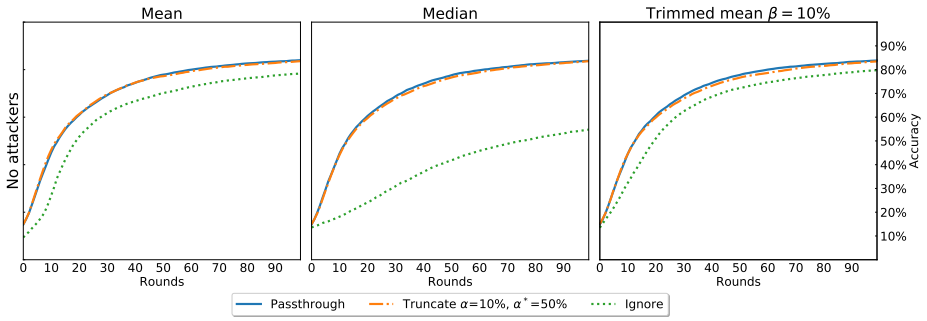

Fig. 5: Accuracy by round without any attackers for the MNIST experiments. Curves correspond to preprocessing procedures and columns correspond to different aggregation methods. It can be seen that our method (dashed orange curve) remains comparable to the properly weighted mean estimators (solid blue 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.

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. The results of our experiments establish that: 1) in the absence of attacks, model convergence is on 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 comparison with the performance of using of all weights in the lack of attacks, whereas without preprocessing, even robust aggregation methods collapse to a performance that is 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 can achieve 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.

As with many Byzantine-robust algorithms, the selection of α has a significant impact on the underlying model and, specifically, on fairness towards clients that hold underrepresented data, which may inadvertently be considered

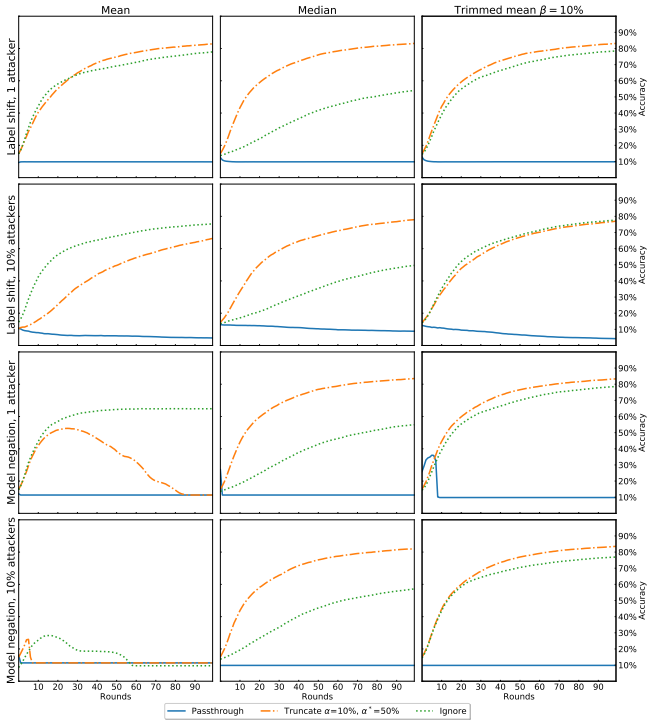

Fig. 6: Accuracy by round under Byzantine attacks for the MNIST experiments. Curves correspond to preprocessing procedures and columns correspond to different aggregation methods. In the first two rows Byzantine clients perform a label shifting attack with one and 10% attackers, respectively. In the 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 remains stable even when there are actual α (=10%) attackers (second and forth rows). In contrast, the same cannot be said for the regular mean aggregator, as can be seen by the sub-optimal accuracy (2nd and 3rd rows) and complete failure to converge (last row) in the leftmost column. 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 5). Moreover, when used in conjunction with robust mean aggregation methods it maintains their robustness properties.

outliers. 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 client sample size altogether.

References

1. Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., Zheng, X.: TensorFlow: Large-scale machine learning on heterogeneous systems (2015), `https://www.tensorflow.org/`, software available from tensorflow.org
2. Alistarh, D., Allen-Zhu, Z., Li, J.: Byzantine stochastic gradient descent. In: Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., Garnett, R. (eds.) Advances in Neural Information Processing Systems 31, pp. 4613–4623. Curran Associates, Inc. (2018), `http://papers.nips.cc/paper/7712-byzantine-stochastic-gradient-descent.pdf`
3. Blanchard, P., El Mhamdi, E.M., Guerraoui, R., Stainer, J.: Machine learning with adversaries: Byzantine tolerant gradient descent. In: Guyon, I., Luxburg, U.V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., Garnett, R. (eds.) Advances in Neural Information Processing Systems 30, pp. 119–129. Curran Associates, Inc. (2017)
4. 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)
5. Caldas, S., Duddu, S.M.K., Wu, P., Li, T., Konečný, J., McMahan, H.B., Smith, V., Talwalkar, A.: Leaf: A benchmark for federated settings (2019)
6. Chen, X., Chen, T., Sun, H., Wu, Z.S., Hong, M.: Distributed training with heterogeneous data: Bridging median- and mean-based algorithms (2019)
7. Chen, Y., Sun, X., Jin, Y.: Communication-efficient federated deep learning with asynchronous model update and temporally weighted aggregation (2019)
8. Chen, Y., Su, L., Xu, J.: Distributed statistical machine learning in adversarial settings: Byzantine gradient descent. Proc. ACM Meas. Anal. Comput. Syst. **1**(2) (Dec 2017). https://doi.org/10.1145/3154503, `https://doi.org/10.1145/3154503`
9. Ghosh, A., Hong, J., Yin, D., Ramchandran, K.: Robust federated learning in a heterogeneous environment. CoRR **abs/1906.06629** (2019), `http://arxiv.org/abs/1906.06629`
10. Haddadpour, F., Mahdavi, M.: On the convergence of local descent methods in federated learning (2019)
11. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Computation **9**, 1735–1780 (1997)
12. 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., Gascón, 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., Konečný, J., Korolova, A., Koushanfar, F., Koyejo, S., Lepoint, T., Liu, Y., Mittal, P., Mohri, M., Nock, R., Özgür, A., Pagh, R., Raykova, M., Qi, H., Ramage, D., Raskar, R., Song, D., Song, W., Stich, S.U., Sun, Z., Suresh, A.T., Tramèr, F., Vepakomma, P., Wang, J., Xiong, L., Xu, Z., Yang, Q., Yu, F.X., Yu, H., Zhao, S.: Advances and open problems in federated learning (2019)
13. Konečnỳ, J., McMahan, B., Ramage, D.: Federated optimization: Distributed optimization beyond the datacenter. arXiv preprint arXiv:1511.03575 (2015)
14. LeCun, Y., Cortes, C., Burges, C.: MNIST handwritten digit database. ATT Labs [Online]. Available: http://yann. lecun. com/exdb/mnist **2** (2010)
15. Li, L., Xu, W., Chen, T., Giannakis, G.B., Ling, Q.: Rsa: Byzantine-robust stochastic aggregation methods for distributed learning from heterogeneous datasets. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 33, pp. 1544–1551 (2019)
16. Li, X., Huang, K., Yang, W., Wang, S., Zhang, Z.: On the convergence of fedavg on non-iid data. arXiv preprint arXiv:1907.02189 (2019)
17. McMahan, H.B., Moore, E., Ramage, D., Hampson, S., y Arcas, B.A.: Communication-efficient learning of deep networks from decentralized data. In: Artificial Intelligence and Statistics. pp. 1273–1282 (2017)
18. Pillutla, K., Kakade, S.M., Harchaoui, Z.: Robust aggregation for federated learning (2019)
19. Reddi, S., Charles, Z., Zaheer, M., Garrett, Z., Rush, K., Konečný, J., Kumar, S., McMahan, H.B.: Adaptive federated optimization (2020)
20. Shakespeare, W.: The Complete Works of William Shakespeare. Complete Works Series, Wordsworth Editions (1996), `https://books.google.co.il/books?id=D02LWkkBxn4C`
21. Yin, D., Chen, Y., Kannan, R., Bartlett, P.: Byzantine-robust distributed learning: Towards optimal statistical rates. In: Dy, J., Krause, A. (eds.) Proceedings of the 35th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 80, pp. 5650–5659. PMLR, Stockholmsmässan, Stockholm Sweden (10–15 Jul 2018)
22. Zhao, Y., Li, M., Lai, L., Suda, N., Civin, D., Chandra, V.: Federated learning with non-iid data (2018)