arXiv:2010.07541v2 [cs.DC] 6 May 2021

Mitigating Byzantine Attacks in Federated Learning

Saurav Prakash [1] **Salman Avestimehr** [1]

Abstract

For mitigating Byzantine behaviors in federated learning (FL), most state-of-the-art approaches, such as Bulyan, tend to leverage the similarity of updates from the benign clients. However, in many practical FL scenarios, data is non-IID across clients, thus the updates received from even the benign clients are quite dissimilar, resulting in poor convergence performance of such similarity based methods. As our main contribution, we propose *DiverseFL* to overcome this challenge in heterogeneous data distribution settings. Particularly, the FL server in DiverseFL computes a *guiding* gradient in every iteration for each client over a small sample of the client's local data that is received only once before start of the training. The server then utilizes a novel *per client* criteria for flagging Byzantine updates, by comparing the corresponding guiding gradient with the client's update, and updates the model using the gradients received from the non-flagged clients. This overcomes the shortcoming of similarity based approaches since the flagging of a client is based on whether its update matches what is expected from its verified sample data (not its similarity to performance of others). As we demonstrate through our experiments involving neural networks, benchmark datasets and popular Byzantine attacks, including a strong backdoor attack for non-IID data, DiverseFL not only performs Byzantine mitigation quite effectively, it *almost matches the performance of Oracle SGD*, where the server knows the identities of the Byzantine clients.

1. Introduction

In the recent past, federated learning (FL) has arose to be a promising approach to enable machine learning using the data distributed at the clients, while keeping the client data private (McMahan et al., 2017; Kairouz et al., 2021). A generic FL framework consists of two major steps – the local SGD computation at each client using a mini-batch from the local data, and, the global model update at the central server using the local updates from the clients. These steps are carried in tandem iteratively until convergence. Fig. 1 illustrates a practical FL use case of a group of hospitals that want to collaborate for learning a predictive model with high accuracy for oncology, while keeping patient data private (Deist, 2019). This distributed training can be orchestrated by a central trusted server such as the NIH.

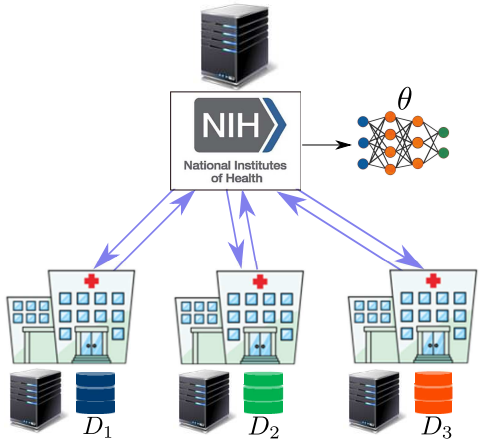

Figure 1. Use case for federated learning: A group of hospitals wish to collaborate to learn a strong predictive model for oncology, while preserving privacy of patient records. Patient data is quite heterogeneous across hospitals due to widely varying characteristics of patients and their surroundings. A trusted coordinator, such as the NIH, can coordinate the training process.

Due to large-scale and decentralized execution, federated learning is vulnerable to software/hardware errors and adversarial attacks. Particularly, some of the clients can become faulty due to software bugs, or even get hacked during training, sending arbitrary or malicious values to the server, thus severely degrading the overall convergence performance. For example, an adversary can reverse the direction of the gradient update at a client by multiplying it by -1, resulting in wrong updates during training. Such faults, where client nodes behave arbitrarily, are called Byzantine faults (Lamport et al., 2019), and it is imperative to make federated learning robust to these behaviors.

[1]University of Southern California. Correspondence to: Saurav Prakash <sauravpr@usc.edu>.

Preprint. Under Review.

To address Byzantine behaviors in FL with IID data, a number of strategies have been proposed recently (Chen et al., 2017; Blanchard et al., 2017; Yin et al., 2018; Su & Xu, 2018; Guerraoui et al., 2018). When data is IID, the gradient updates from the benign clients tend to be distributed around the true gradient, while the gradient updates from the Byzantine clients can be arbitrary and are thus handled by applying robust estimation techniques for aggregating the client updates. For example, (Yin et al., 2018) proposes element-wise median and marginal trimmed operations on the gradient updates from the clients, while in (Su & Xu, 2018), authors propose an iterative filtering for robust gradient aggregation. While (Chen et al., 2017) proposes a geometric median rule for the strongly convex setting, (Blanchard et al., 2017) proposes a general but computationally expensive algorithm called Krum, that in each iteration selects the gradient update that in essence, has the minimum sum of distances from the nearest gradient updates.

In (Guerraoui et al., 2018), the authors observe that in high dimensions, which is the case in large neural networks, when the distance criteria is relying completely on ℓ_p norms, a Byzantine update can be made to remain close to honest clients and yet have a single dimension poisoned by significantly amplifying its magnitude, thus circumventing Krum. To mitigate this hidden attack, (Guerraoui et al., 2018) proposes Bulyan, that first finds a subset of nodes using Krum recursively, and then applies an element-wise trimmed mean on the remaining updates to filter out the high magnitude values. Another stealthy attack is proposed in (Baruch et al., 2019), which circumvents existing state-of-the-art defense approaches such as Bulyan. The attack, however, is designed for homogeneous data distribution across devices and is thus not applicable to FL settings with non-IID data.

In many practical FL settings, data is non-IID across clients. For example, as highlighted in Fig. 1, a key aspect of medical data is data heterogeneity because patient characteristics often vary depending on a variety of factors such as age, climate and air quality. When data is non-IID, achieving Byzantine resiliency becomes even more challenging as the updates from even the benign clients are quite dissimilar, thus degrading the performance of prior Byzantine resilient approaches in IID setting. Some recent works such as (Chen et al., 2019; Li et al., 2019) deal with heterogeneous data distribution in federated learning. In (Li et al., 2019), the authors propose to reformulate the optimization problem as a consensus based optimization problem, and incorporate a regularization term into the objective function for mitigating Byzantines. However, the proposed algorithm requires strong convexity for convergence. In (Chen et al., 2019), the authors propose a noisy median approach to improve performance of the median algorithm (proposed in (Yin et al., 2018)) in heterogeneous data settings in the absence of Byzantine behaviors. The main idea is that each client computes gradient over its entire local dataset and adds unimodal and symmetric noise to it so that the dimensional median of the noisy gradients has better convergence properties. In our experiments, we demonstrate that noisy median approach improves convergence performance with non-IID data when training with modest neural networks and MNIST dataset even when each client uses a minibatch for computing gradient. However, performance of noisy median degrades significantly for large neural networks (VGG11) and minibatch training with more complex datasets (CIFAR10 and CIFAR100), since the noise due to minibatch subsampling overpowers the benefits of adding unimodal and symmetric random noise to gradient updates in the noisy median approach.

For federated learning with non-IID data distribution across clients, (Bagdasaryan et al., 2020) proposes a strong model replacement attack that can control the training process such that the model performs well on a backdoor task chosen by the attacker while simultaneously having good performance on the main federated learning task. The proposed attack leverages the fact that gradient updates in non-IID FL setting are quite diverse, hence malicious clients can scale their updates suitably so as to replace the true updated model with a stealth model, without being detected by prior Byzantine resilient approaches.

Our contributions: For mitigating Byzantine behaviors in federated learning with non-IID data distribution, we propose *DiverseFL* that utilizes a *per client* criteria for filtering out Byzantine updates. Before training starts, each client shares a small, uniformly sampled subset of its local data with the FL server. During training, for each client, the server computes a *guiding* gradient over the client sample, and leverages this to estimate whether the corresponding client is Byzantine. The main principle is that the guiding gradient associated with a client is similar to the stochastic gradient update from the client if it is benign. Similarly, an arbitrary update from a Byzantine client is quite different from its associated guiding gradient. Based on these ideas, we propose a novel criteria for Byzantine mitigation that works on a per client basis. Specifically, the server flags a client as Byzantine if any of the our proposed similarity conditions are violated – (i) the dot product between the guiding gradient and the client update is greater than 0, (ii) the ratio of the Euclidean norms of the two are within a pre-defined range. The server then aggregates the gradient updates from the non-flagged clients and updates the model.

In our experiments in Section 4, where we consider both softmax regression (convex and smooth optimization) and neural network training (non-convex and non-smooth optimization) with benchmark datasets, we demonstrate that even when each client shares as low as 3% of its local data with the FL server, leveraging the computational resource of

the server for computing and utilizing the guiding gradients for detecting Byzantine clients results in a significant improvement in convergence performance. In fact, DiverseFL almost achieves the convergence performance of Orcale SGD, wherein the FL server knows the identities of the Byzantine clients. We also note that since clients share only small samples of local data with the server, Byzantine clients that share corrupted data with the server can be detected and removed before the start of training through manual sample analysis (Holzinger, 2016) by using expert knowledge. For example, in the FL use case illustrated in Fig. 1, a group of experts at the NIH can check for corrupted samples. As demonstrated in our experiments, this one time pre-processing effort can result in significant improvement in model accuracy, which is a critical goal in healthcare.

We would also like to point out that our focus in the above contributions has been to understand how little bit of sample sharing from the clients can significantly improve performance of federated learning in presence of Byzantine faults. However, addressing the privacy and security of these shared data samples is a very exciting problem for future directions and there are possible solutions to address this issue. One approach is to leverage secure environments such as SGX (Costan et al., 2016; McKeen et al., 2013), so that guiding gradient executions at the server are done in trusted environments so that the server does not actually see the data. Another approach for this could be secret sharing of client data samples with two servers and relying on secure two-party computation (Pinkas et al., 2009). Furthermore, for verification of labels of those samples, one can rely on a generic pre-trained model as a sanity check, and this sample verification process can be also carried out in SGX so that data samples are not revealed to the FL server.

As a final remark, we point out that the idea of leveraging shared data for improving convergence performance in federated learning with heterogeneous data was originally proposed in (Zhao et al., 2018). However, as opposed to our work where each client shares a small sample of local data and offloads a small computational task to the server for guiding gradient computation, (Zhao et al., 2018) does not perform any gradient computation at the server. Rather, before training starts, the server shares a random sample of pooled data with each client. This causes homogenization of the data across clients and results in improved convergence. Furthermore, (Zhao et al., 2018) does not consider the presence of Byzantine faults in federated learning.

2. Problem Setup

In this section, we describe stochastic gradient descent (SGD) in the context of federated learning in the absence of Byzantine faults, and present the Byzantine fault model.

2.1. Federated Learning with SGD

Federated learning setup comprises of n client nodes (workers) that are connected to a central server (master). We consider supervised machine learning, and assume that there is an underlying sample space for the data which is denoted by $\Omega = X \times Y$, where X denotes the space of feature vectors while Y denotes the space of corresponding labels. For $j \in [n] = \{1, \ldots, n\}$, let $\mathcal{D}_j$ denote the local data distribution over Ω at client j, and let $\zeta_j \sim \mathcal{D}_j$ denote the random variable for the feature-label tuple drawn from $\mathcal{D}_j$. In practice, the classical assumption that data is IID across clients does not hold (Zhao et al., 2018). Therefore, $\mathcal{D}_i$ can be different from $\mathcal{D}_j$ for $i, j \in [n], i \neq j$. Let $f_j(\theta)$ be the population loss function for client j as defined below:

$$f_j(\theta) \triangleq \mathbb{E}_{\zeta_j}[l(\theta; \zeta_j)], \tag{1}$$

where $l(\theta; (x_j, y_j)) \in \mathbb{R}$ denotes the predictive loss function for model parameter $\theta \in \mathbb{R}^d$ and a given realization (x_j, y_j) of ζ_j. We focus on solving the following problem:

$$\theta^* = \arg\min_{\theta \in \mathbb{R}^d} \frac{1}{n} \sum_{j=1}^{n} f_j(\theta) + \frac{\lambda}{2} \|\theta\|_2^2, \tag{2}$$

The solution to (2) is obtained by using an iterative training procedure that involves two key steps. Consider a general iteration r, $r \in \{1, \ldots r_{\max}\}$, where $r_{\max}$ denotes the hyper-parameter for the maximum number of iterations. In the first step, each client carries out a mini-batch stochastic gradient descent (SGD) computation. Specifically, j-th client receives the current model θ^{r-1} from the master, and samples a batch $\mathcal{M}_j^r = \{(x_{j,k}, y_{j,k})\}_{k=1}^m$ of data points from $\mathcal{D}_j$. It then obtains an estimate of the gradient $\nabla_\theta f_j(\theta^{r-1})$ of the local population loss function defined in (1) by computing the gradient of the empirical loss over $\mathcal{M}_j^r$ as follows:

$$g_j^r = \frac{1}{m} \sum_{k=1}^{m} \nabla_\theta l\left(\theta^{r-1}; (x_{j,k}, y_{j,k})\right). \tag{3}$$

In the second step, the master receives the gradients from the clients and updates the model. Specifically, it carries out the following computation:

$$\theta^r = \theta^{r-1} - \mu^r \left(g^r + \lambda \theta^{r-1}\right), \tag{4}$$

where $g^r = \frac{1}{n} \sum_{j=1}^{n} g_j^r$, and μ^r is the learning rate. Steps (3) and (4) are executed in tandem for $r = r_{\max}$ iterations.

2.2. Byzantine Fault Model

Due to decentralized and distributed implementation of federated learning, clients are vulnerable to Byzantine faults (arbitrary errors) arising out of hardware/software failures or hackers causing coordinated and malicious attacks. As a result, Byzantine clients can upload arbitrary updates to the

master, significantly degrading the learning performance. For iteration r, let $z_j^r \in \mathbb{R}^d$ denote the update uploaded by client j, $j \in \{1, \ldots, n\}$. We have the following:

$$z_j^r = \begin{cases} g_j^r & \text{if } j \in \mathcal{N}^r, \\ * & \text{if } j \in \mathcal{B}^r, \end{cases} \tag{5}$$

where for r-th iteration, $\mathcal{B}^r \subseteq \{1, \ldots, n\}$ denotes the set of Byzantine clients and $\mathcal{N}^r = \{1, \ldots, n\} \backslash \mathcal{B}^r$ denotes the set of normal clients, while $*$ denotes that z_j^r can be an arbitrary vector in $\mathbb{R}^d$.

Furthermore, we note that the set of Byzantine clients (and likewise the set of benign clients) can vary during training, for example, a client exhibiting Byzantine behaviors due to some underlying hardware or software failures may recover after a few iterations. Hence, for $r \in \{1, \ldots, r_{\max}-1\}$, $\mathcal{B}^r$ and $\mathcal{B}^{r+1}$ are not necessarily equal.

3. DiverseFL

We now describe our proposed scheme DiverseFL in which Byzantine mitigation is based on a per-client criteria, and not on similarity of performance of benign clients. At the start of training, client $j \in [n]$ draws a sample $\mathcal{M}_j^0 = \{(x_{j,k}, y_{j,k})\}_{k=1}^s$ from its local data distribution $\mathcal{D}_j$ and shares it with the master. Here, $s = |\mathcal{M}_j^0|$ denotes the size of each client's sample.

Training in DiverseFL proceeds as follows. During r-th iteration, client $j \in \mathcal{N}^r$ receives the current model θ^{r-1} from the master and computes the gradient g_j^r over a mini-batch sample $\mathcal{M}_j^r$ of size m using (3), and uploads $z_j^r = g_j^r$ to the master. Clients in $\mathcal{B}^r$ can upload arbitrary vectors. The server computes the gradient of the empirical loss corresponding to the data sample $\mathcal{M}_j^0 = \{(x_{j,k}, y_{j,k})\}_{k=1}^s$ for each client $j \in [n]$ as follows:

$$\widetilde{g}_j^r = \frac{1}{s} \sum_{k=1}^{s} \nabla_\theta l\left(\theta^{r-1}; (x_{j,k}, y_{j,k})\right). \tag{6}$$

The master then estimates whether client $j \in [n]$ is Byzantine or not based on the extent of *similarity between* z_j^r *and* $\widetilde{g}_j^r$, using $\widetilde{g}_j^r$ as a surrogate for g_j^r. It consider the following two similarity metrics:

$$\text{Direction Similarity} : C_1 = \text{sign}(\widetilde{g}_j^r \cdot z_j^r), \tag{7}$$

$$\text{Length Similarity} : C_2 = \frac{\|z_j^r\|_2}{\|\widetilde{g}_j^r\|_2}. \tag{8}$$

The main idea here is that $\widetilde{g}_j^r$ approximates the SGD update g_j^r of a benign client. Therefore, the master can exploit the similarity between $\widetilde{g}_j^r$ and $z_j^r = g_j^r$ when $j \in \mathcal{N}^r$, and the dissimilarity between $\widetilde{g}_j^r$ and z_j^r when $j \in \mathcal{B}^r$. Thus, when $C_1 > 0$, it suggests that $-z_j^r$ is *approximately* a descent direction for client j. Similarly, when $C_2 \sim 1$, it suggests that the norms $\|z_j^r\|_2$ and $\|g_j^r\|_2$ are *approximately* equal, while very large or very small values of C_2 suggest large deviation between z_j^r and g_j^r. These arguments motivate the following two conditions that the master checks for each client $j \in [n]$:

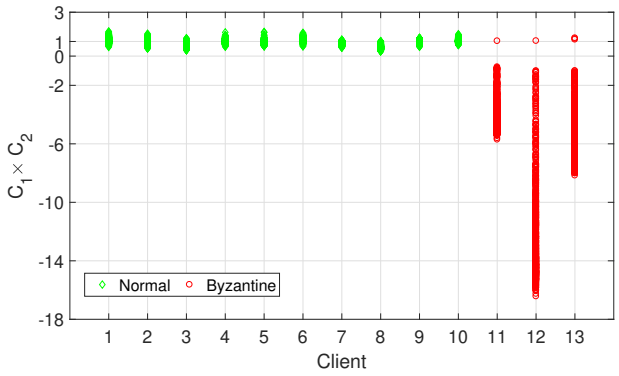

Figure 2. Visualization of the similarity criteria presented in (7) and (8). We consider 1000 training iterations of oracle SGD for the setting with 23 clients, MNIST dataset and neural network training with label flip Byzantine attack, as described in Section 4.2. 5 out of the 23 clients are selected to be Byzantine for the training process. For brevity, we plot $C_1 \times C_2$ for only 10 benign clients and 3 Byzantine clients, while the results for the remaining clients are similar. The values of $C_1 \times C_2$ in the 1000 iterations are put in red for Byzantine clients, and in green, for benign clients. As can be clearly observed, for benign clients, $C_1 > 0$ exclusively and C_2 is concentrated around 1. For Byzantine clients, $C_1 < 0$ in almost all iterations, and C_2 varies significantly.

$$\text{Condition 1} : C_1 > 0, \tag{9}$$

$$\text{Condition 2} : \epsilon_1 < C_2 < \epsilon_2, \tag{10}$$

where ϵ_1 and ϵ_2 are hyperparameters in DiverseFL. The master flags client j as a Byzantine node if either of the above two conditions is not satisfied. Let $\widetilde{\mathcal{B}}^r$ denote the set of Byzantine nodes as estimated using the above criteria, and likewise let $\widetilde{\mathcal{N}}^r = [n] \backslash \widetilde{\mathcal{B}}^r$ denote the estimated set of benign clients. The master takes the following model update step:

$$\theta^r = \theta^{r-1} - \mu^r \left(\frac{1}{|\widetilde{\mathcal{N}}^r|} \sum_{j \in \widetilde{\mathcal{N}}^r} z_j^r + \lambda \theta^{r-1} \right). \tag{11}$$

A summary of DiverseFL in provided Algorithm 1.

To illustrate the effectiveness of these ideas in practice, we consider the product $C_1 \times C_2$ of the metrics in (7) and (8), and plot its variation for different clients across iterations. For this, we consider the setting of 23 clients and neural network training with MNIST dataset, in the presence of label flip attack, as described in Section 4.2. Data is distributed non-IID, and 5 out of the 23 clients are selected to be Byzantine throughout the training procedure. We consider the

Algorithm 1 DiverseFL

Input: Hyperparameters ϵ_1, ϵ_2, λ, $r_{\max}$, $\{\mu^r\}_{r=1}^{r_{\max}}$, m, s
Output: Trained model $\theta^{r_{\max}}$
Training at Master:

Obtain data sample $\mathcal{M}_j^0$, from each client $j\in[n]$
Initialize model θ^0
for iteration $r = 1, \ldots, r_{\max}$ **do**
  Send θ^{r-1} to all clients $j\in[n]$
  Compute sample gradient $\widetilde{g}_j^r$ for each client $j\in[n]$
  Receive the update z_j^r for each client $j\in[n]$
  $\widetilde{\mathcal{B}}^r \leftarrow \phi$, $\widetilde{\mathcal{N}}^r \leftarrow \phi$
  for $j\in[n]$ **do**
    Evaluate C_1 using (9) and C_2 using (10)
    if ($C_1>0$ and $\epsilon_1<C_2<\epsilon_2$) **then**
      $\widetilde{\mathcal{N}}^r \leftarrow \widetilde{\mathcal{N}}^r \cup \{j\}$
    else
      $\widetilde{\mathcal{B}}^r \leftarrow \widetilde{\mathcal{B}}^r \cup \{j\}$
    end if
  end for
  Compute new model θ^r using (11)
end for

Training at Client $j\in[n]$**:**

Obtain random sample $\mathcal{M}_j^0$, $|\mathcal{M}_j^0|=s$ from data distribution $\mathcal{D}_j$, send $\mathcal{M}_j^0$ to master
for iteration $r = 1, \ldots, r_{\max}$ **do**
  Receive θ^{r-1} from master
  if j is benign **then**
    Obtain a random sample $\mathcal{M}_j^r$ from $\mathcal{D}_j$,
    Compute g_j^r using (3)
    $z_j^r \leftarrow g_j^r$
  else
    $z_j^r \leftarrow *$
  end if
  Send z_j^r to master
end for

oracle SGD algorithm in which the master only aggregates all the benign client updates for updating the model. Further details of the experiment are deferred to Section 4.2.

In Fig. 2, we plot the product $C_1 \times C_2$ for 1000 SGD iterations for 10 different benign clients and 3 different Byzantine clients, where green color marker denotes that client is benign, while red color marker denotes that client is Byzantine. As can be seen from Fig. 2, $C_1>0$ exclusively and C_2 is concentrated around 1 for each client whenever it is benign. When a client is Byzantine, $C_1<0$ *almost* exclusively, and there is a large variation of C_2 during training. For example, out of the 1000 training iterations, $C_1>0$ for only 1 iteration for both client 11 and client 12, and for 3 iterations for client 13. Similar results were observed for the other clients. Therefore, condition 1 presented in (9) is justified and is quite useful in mitigating Byzantine faults. Condition 2 presented in (10) is complementary to condition 1, as it helps to keep the deviations from the true updates low, thus helping the server to filter Byzantine updates in scenarios such as those in which g_j^r is simply scaled with a large absolute value. As we demonstrate in experiments in Section 4, the proposed similarity criteria results in superior performance of DiverseFL.

4. Experiments

Our experiments are broadly divided into two parts. We first focus on softmax regression with cross-entropy loss with MNIST, which is essentially a convex optimization problem. Thereafter, to demonstrate the potential of DiverseFL in achieving state-of-the-art performance in practice, we consider neural network training with cross-entropy loss with MNIST/CIFAR10/CIFAR100, which essentially involves non-convex and non-smooth loss functions due to variety of non-linearities such as ReLUs.

4.1. Softmax Regression

Setting: We consider n=23 clients and use MNIST dataset that has 10 classes. To simulate non-IID data, we sort the training data as per class, partition the sorted data into n subsets, and assign each client 1 partition. Furthermore, we set f=5 clients as Byzantines throughout training.

In the following, we summarize the prior benchmark schemes that we simulate for comparison.
Oracle SGD (abbr. OracleSGD): It is the ideal scheme wherein the master updates the model using the aggregate of gradients from only the benign clients.
Median (Yin et al., 2018): The server takes an element-wise median of all the received updates from the clients.
Bulyan (Guerraoui et al., 2018): The server applies Krum recursively to remove a set of $2f$ possibly Byzantine clients, and then applies an element-wise trimmed mean operation to filter out $2f$ entries in each dimension that are farthest from the dimensional median. Both Krum and the trimmed mean algorithms are included in the supplementary.
Noisy Median (Chen et al., 2019) (abbr. N-Median): Each benign client adds random Gaussian noise with mean 0 and standard deviation σ_{N} to each entry in its gradient update, while the server takes an element-wise median of all the received updates from the clients.
RSA (Li et al., 2019): This is a consensus based algorithm that incorporates a novel ℓ_1-norm regularization term into the objective function with hyperparameter δ for mitigating Byzantines. Also, clients upload copies of their local models to the server instead of sharing their gradient update.
The formal detailed descriptions of the above approaches are included in the supplementary.

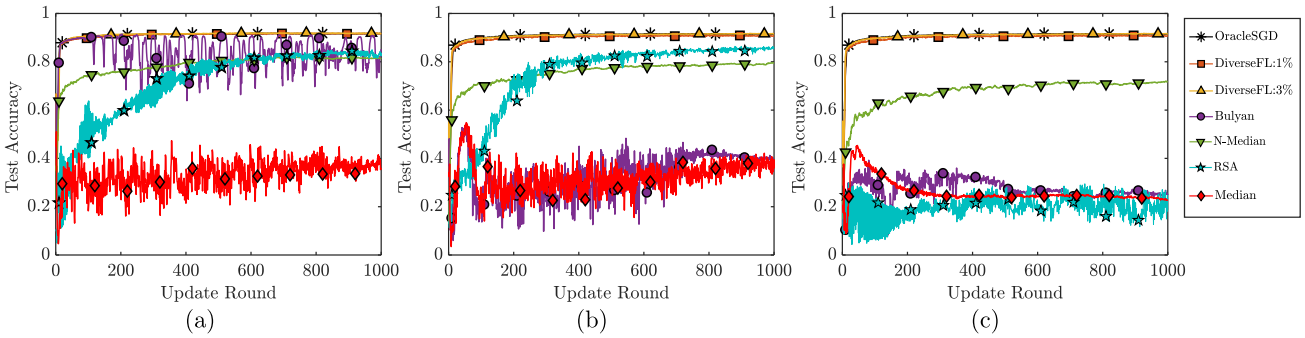

Figure 3. Top-1 accuracy for softmax regression with MNIST: (a) No Attack; (b) Gaussian Attack; (c) Sign Flip Attack.

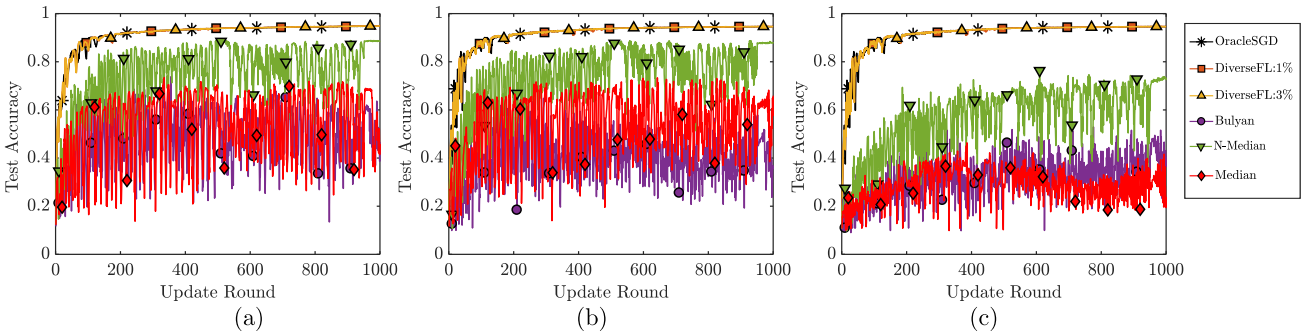

Figure 4. Top-1 accuracy for neural network training with MNIST: (a) No Attack; (b) Gaussian Attack; (c) Sign Flip Attack.

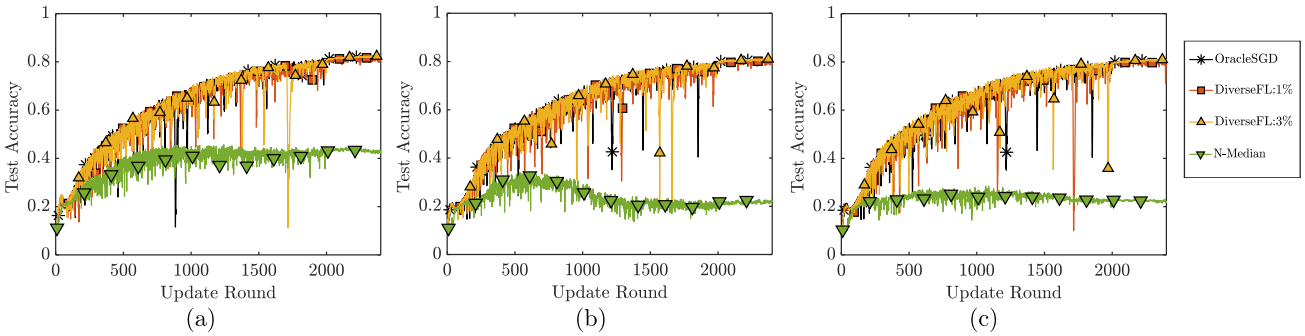

Figure 5. Top-1 accuracy for neural network training with CIFAR10: (a) No Attack; (b) Gaussian Attack; (c) Sign Flip Attack.

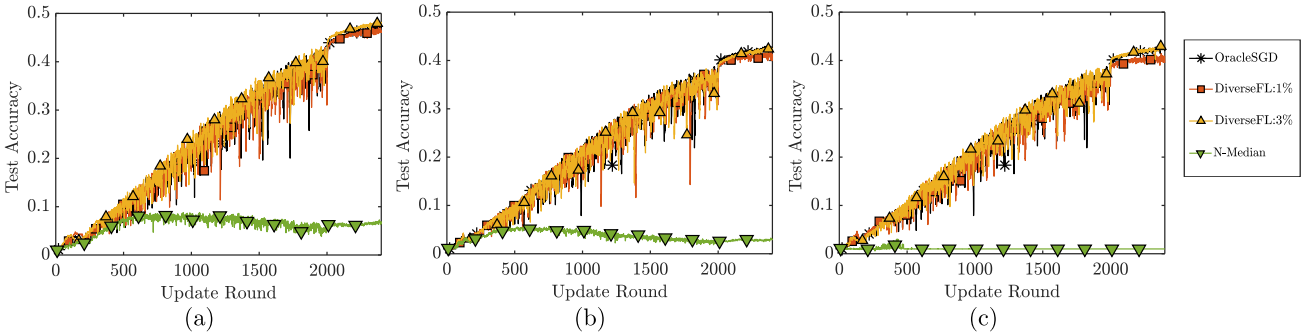

Figure 6. Top-1 accuracy for neural network training with CIFAR100: (a) No Attack; (b) Gaussian Attack; (c) Sign Flip Attack.

For a Byzantine client j in iteration r, we consider four popular Byzantine attack models as described next.

Gaussian: The message to be uploaded is set to z_j^r, where the elements follow a Gaussian distribution $\sim \mathcal{N}(0, \sigma_G^2)$.

Sign Flip: The sign of each entry of the update is flipped before uploading to the server.

Same Value: The message to be uploaded is set to $z_j = \sigma_S \mathbf{1}$, where $\mathbf{1} \in \mathbb{R}^d$ is an all-one vector.
Label Flip: Class label c of each data point in the training mini-batch $\mathcal{M}_j^r$ is flipped to class label $c_n - c$, where c_n is the number of classes.

We choose a local mini-batch size of m=300, and perform training for $r_{\max}$=1000 iterations with a decreasing learning rate of $\mu^{(r)}$=$0.001/\sqrt{r}$, similar to the learning rate decay criteria in (Li et al., 2019). We set λ=0.0067 for the ℓ_2 regularization. We use δ=0.25 for RSA, σ_N=100 for N-Median, and use sample size s=30 and thresholds for ratio of norms (ϵ_1, ϵ_2)=$(0.5, 2)$ for DiverseFL. Model parameters are initialized to 0. For Gaussian and same value attacks, we set σ_G=σ_S=10^4.

Results: Fig. 3(a) demonstrates that even in the absence of Byzantine attacks, the performance of DiverseFL dominates all prior Byzantine-resilient benchmark approaches, and almost matches the performance of Oracle SGD. As expected, Median performs quite poorly as the gradients from the benign clients are quite dissimilar, so an element-wise median of even the benign client updates does not perform well. N-Median improves the convergence performance of Median by addition of Gaussian noise by the benign clients. Furthermore, Bulyan provides a reasonable performance for softmax regression, although the performance ends up oscillating withing a large range. RSA achieves stable performance almost matching the performance of N-Median.

In Fig. 3(b) and 3(c), we present the results of Byzantine-resiliency against Gaussian and sign Flip attacks, while the results for same value and label flip attacks have been included in the supplementary. Expectedly, along with Median, Bulyan performs quite poorly, since its distance based filtering followed by its trimmed mean operation is not able to distinguish between dissimilar benign updates and the updates from the Byzantine clients. Noisy Median is quite resilient to different attacks in softmax regression, although its performance is worse for sign flip attack than for Gaussian attack. RSA performs well in presence of Gaussian attack but performs quite poorly in presence of sign flip attack. DiverseFL on the other hand, is quite stable, and achieves the performance of OracleSGD for all these attacks. Furthermore, we observe from Fig. 3 that sharing of just 1% of client data with servers is enough for achieving superior convergence and Byzantine-resiliency in DiverseFL. Lastly, we note that although original work of (Chen et al., 2019) did not evaluate performance of N-Median in presence of Byzantine attacks, our simulation results demonstrate that it provides reasonable convergence performance in the presence of Byzantine faults, which can be of independent interest.

4.2. Neural Networks

We demonstrate superior convergence and Byzantine-resiliency of DiverseFL in neural network training. In addition to the four non-targeted Byzantine attacks described in Section 4.1, we simulate the strong targeted backdoor attack proposed in (Bagdasaryan et al., 2020) for the non-IID dataset distribution, where the goal is to control the training process so that the model performs well on a subtask chosen by the attacker while having good performance on the original federated learning task. We first discuss the experiments for the non-targeted attacks, and then present the the experiments for the targeted backdoor attack.

Setting (non-targeted attacks): Similar to the softmax regression experiments, we have a setting with n=23 clients out of which 5 are Byzantine. We consider three different datasets – MNIST, CIFAR10, and CIFAR100 (LeCun & Cortes, 2010; Krizhevsky et al., 2009). While MNIST and CIFAR10 have 10 classes each, CIFAR100 has 100 classes. For each dataset, the training data is sorted as per class and then partitioned into n subsets, and each subset is assigned to a different client. For MNIST, we use a neural network model with two fully connected layers, each with 200 neurons. For CIFAR10 and CIFAR100, we use VGG network (Simonyan & Zisserman, 2014), specifically VGG11, which has a total of 8 convolutional layers, and 3 fully connected layers. The full architecture details are provided in the supplementary. Additionally, we note that since batch normalization layers have been found to be a bottleneck for neural network training from decentralized and heterogeneous data (Ioffe, 2017; Hsieh et al., 2020), we train with modified VGG11 network wherein we replace all the batch normalization layers with group normalization layers (Wu & He, 2018).

For each scenario, local training batch size is 10% of the local dataset, regularization λ=0.0005, and for DiverseFL, we use (ϵ_1, ϵ_2)=$(0.5, 2)$ and two sampling size scenarios of 1% and 3% of the local dataset. For Gaussian and same values attacks for each dataset, we set σ_G=σ_S=10. For MNIST, we set $r_{\max}$ to 1000, use an initial learning rate of 0.06 with step decay of 0.5 at iterations 500 and 950, and compare DiverseFL with OracleSGD, Median, Bulyan and N-Median, setting σ_N=0.01 for N-median. For CIFAR10 and CIFAR100, warmup is used for the first 1000 iterations, increasing the learning rate linearly from 0.05 to 0.1. Furthermore, total iterations is set to $r_{\max}$=2400 and learning rate is stepped down by a factor of 0.4 at iteration 2000 and σ_N=0.1 for N-median. As we demonstrate in the results presented next, performance of Median and Bulyan degrades significantly even for neural network training with MNIST. Furthermore, experiments with CIFAR10 and CIFAR100 are quite time consuming, particularly those involving Bulyan. Hence, for CIFAR10 and CIFAR100,

we focus on comparison of DiverseFL with OracleSGD and N-Median only in our results. In supplementary, we present additional results involving Median and Bulyan for CIFAR10 using a smaller neural network. Lastly, we note that RSA is excluded from all neural network simulations as it is primarily designed for a convex loss function, and did not converge in any of our neural network experiments.

Results (non-targeted attacks): Fig. 4, Fig. 5 and Fig. 6 respectively illustrate the results for MNIST, CIFAR10 and CIFAR100. The results for same value and label flip attacks are included in the supplementary. For MNIST, no significant improvement in convergence performance is observed when the sampling size for DiverseFL is increased from 1% to 3%. DiverseFL outperforms prior approaches by significant margins in all cases of Byzantine attacks, and almost matches up to the performance of OracleSGD. Bulyan performs poorly for neural network training even in absence of Byzantine attacks. For CIFAR10, DiverseFL with both 1% and 3% sampling sizes perform quite close to OracleSGD, with the 3% case performing slightly better. For CIFAR100, the impact of sampling size is noticeable, mainly because of large number of classes in the the dataset. Nevertheless, even with a small sampling size of 3%, DiverseFL achieves close to the best performance of OracleSGD. It is an interesting future direction to quantify the relationship between sample size and number of classes. Furthermore, we note that the margin of convergence performance between OracleSGD and N-Median is much larger for CIFAR10 than for MNIST, due to greater dataset complexity and larger neural networks, that introduce increased subsampling noise during training, dominating the benefits of adding Gaussian noise at the benign clients. For CIFAR100, each client exclusively has data points from roughly 4 different classes, causing the sampling noise to increase much further, therefore degrading the performance of N-Median significantly.

Setting (targeted backdoor attack): For backdoor attack, we focus on MNIST and CIFAR10 datasets, using the same neural network architectures as in the non-targeted attacks. For MNIST, the backdoor task is to label images with digit 3 as digit 4, and the main task is to correctly detect the remaining classes. For CIFAR10, the backdoor task is to classify all images from 'frog' class as 'ship', and the main task is to correctly detect the remaining classes. We assume that all the clients owning the backdoor images are Byzantine, while remaining are all benign. As proposed in (Bagdasaryan et al., 2020), the backdoor attack involves carrying out a local update using a mix of normal and backdoor images, so that accuracy on the main training task is not affected. Hence, we assign data subsets to the clients such that each client is assigned data points from at least two different classes, thus ensuring that a random sample at any Byzantine client has $\sim 50\%$ main task images. We use model replacement attack as proposed in (Bagdasaryan et al., 2020), and scale the local gradient update at a Byzantine client by a factor of 7 to ensure the update is not cancelled during the averaging.

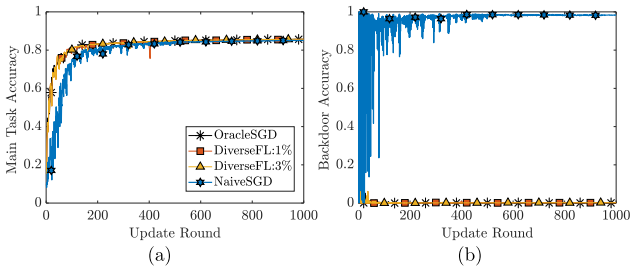

Figure 7. Results for Backdoor Attack for MNIST: (a) Main Task Accuracy; (b) Backdoor Accuracy.

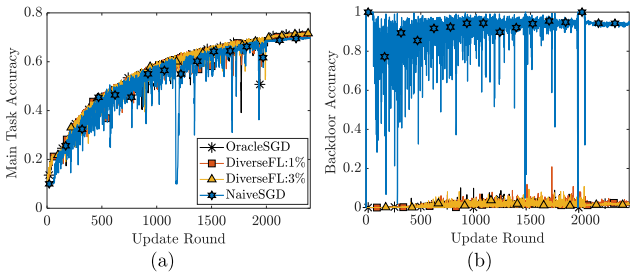

Figure 8. Results for Backdoor Attack for CIFAR10: (a) Main Task Accuracy; (b) Backdoor Accuracy.

Results (targeted backdoor attack): Fig. 7(a) Fig. 8(a) illustrate the accuracy of the trained model on the main task for MNIST and CIFAR10, while Fig. 7(b) Fig. 8(b) demonstrate the accuracy on the backdoor task, where we have also included the performance of naive aggregation of client updates at the server. As expected, both naive SGD and DiverseFL almost achieve the main task convergence performance of OracleSGD. However, naive SGD achieves a very high accuracy on the backdoor task, while DiverseFL successfully mitigates the Byzantine clients. Furthermore, increasing the sample size from 1% to 3% does not provide any noticeable improvement in performance for DiverseFL.

5. Conclusions

We propose DiverseFL that applies a per client criteria for filtering Byzantine clients in federated learning, as opposed to leveraging the similarity of updates from the benign clients. Our proposal demonstrates how small amounts of data sharing by the clients can dramatically improve the model accuracy and Byzantine-resiliency of federated learning with heterogeneous data distribution across clients, a typical scenario in oncology research from patient data at specialized cancer hospitals. We benchmark the superior performance of DiverseFL with neural network training and a variety of Byzantine attacks, including a strong targeted backdoor attack for heterogeneous data distribution at the clients.

References

Bagdasaryan, E., Veit, A., Hua, Y., Estrin, D., and Shmatikov, V. How to backdoor federated learning. In *International Conference on Artificial Intelligence and Statistics*, pp. 2938–2948. PMLR, 2020.

Baruch, G., Baruch, M., and Goldberg, Y. A little is enough: Circumventing defenses for distributed learning. In *Advances in Neural Information Processing Systems*, pp. 8635–8645, 2019.

Blanchard, P., Guerraoui, R., Stainer, J., et al. Machine learning with adversaries: Byzantine tolerant gradient descent. In *Advances in Neural Information Processing Systems*, pp. 119–129, 2017.

Chen, X., Chen, T., Sun, H., Wu, Z. S., and Hong, M. Distributed training with heterogeneous data: Bridging median and mean based algorithms. *arXiv preprint arXiv:1906.01736*, 2019.

Chen, Y., Su, L., and Xu, J. Distributed statistical machine learning in adversarial settings: Byzantine gradient descent. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 1(2):1–25, 2017.

Costan, V., Lebedev, I., and Devadas, S. Sanctum: Minimal hardware extensions for strong software isolation. In *25th {USENIX} Security Symposium ({USENIX} Security 16)*, pp. 857–874, 2016.

Deist, T. M. Distributed learning and prediction modelling in radiation oncology. 2019.

Guerraoui, R., Rouault, S., et al. The hidden vulnerability of distributed learning in Byzantium. In *International Conference on Machine Learning*, pp. 3521–3530. PMLR, 2018.

Holzinger, A. Interactive machine learning for health informatics: when do we need the human-in-the-loop? *Brain Informatics*, 3(2):119–131, 2016.

Hsieh, K., Phanishayee, A., Mutlu, O., and Gibbons, P. B. The non-iid data quagmire of decentralized machine learning. *International Conference on Machine Learning*, 2020.

Ioffe, S. Batch renormalization: Towards reducing minibatch dependence in batch-normalized models. In *Advances in neural information processing systems*, pp. 1945–1953, 2017.

Kairouz, P., McMahan, H. B., et al. Advances and open problems in federated learning. *Foundations and Trends® in Machine Learning*, 14(1), 2021.

Krizhevsky, A., Nair, V., and Hinton, G. Learning multiple layers of features from tiny images. 2009. URL https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf.

Lamport, L., Shostak, R., and Pease, M. The Byzantine generals problem. In *Concurrency: the Works of Leslie Lamport*, pp. 203–226. 2019.

LeCun, Y. and Cortes, C. MNIST handwritten digit database. 2010. URL http://yann.lecun.com/exdb/mnist/.

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, 2019.

McKeen, F., Alexandrovich, I., Berenzon, A., Rozas, C. V., Shafi, H., Shanbhogue, V., and Savagaonkar, U. R. Innovative instructions and software model for isolated execution. *Hasp@ isca*, 10(1), 2013.

McMahan, H. B., Moore, E., Ramage, D., Hampson, S., et al. Communication-efficient learning of deep networks from decentralized data. *AISTATS*, 2017.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, pp. 8024–8035, 2019.

Pinkas, B., Schneider, T., Smart, N. P., and Williams, S. C. Secure two-party computation is practical. In *International conference on the theory and application of cryptology and information security*, pp. 250–267. Springer, 2009.

Simonyan, K. and Zisserman, A. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Su, L. and Xu, J. Securing distributed machine learning in high dimensions. *arXiv preprint arXiv:1804.10140*, 2018.

Wu, Y. and He, K. Group normalization. In *Proceedings of the European conference on computer vision (ECCV)*, pp. 3–19, 2018.

Xie, C., Koyejo, O., and Gupta, I. Generalized Byzantine-tolerant SGD. *arXiv preprint arXiv:1802.10116*, 2018.

Yin, D., Chen, Y., Ramchandran, K., and Bartlett, P. Byzantine-robust distributed learning: Towards optimal statistical rates. *ICML*, 2018.

Zhao, Y., Li, M., Lai, L., Suda, N., Civin, D., and Chandra, V. Federated learning with non-iid data. *arXiv preprint arXiv:1806.00582*, 2018.

Supplementary Overview

In the following, we summarize the content of this supplementary document:
1. In Section A, we formally describe the prior baselines that we use in our experiments for performance comparison with DiverseFL – Median, Bulyan, noisy Median, and RSA. As Bulyan involves both Krum and an element-wise trimmed mean operation, we describe them as well.
2. In Section B, we present the complete results for the settings presented in our main submission, including the results for label flip and same value attacks for the settings considered in Section 4 of our main submission. For quick reference, we also include the results that have been provided in the main submission, i.e. the results with no attack, Gaussian attack, and sign flip attack.
3. In Section C, we provide results of neural network training over CIFAR10 using a smaller neural network than VGG11, in order to also compare with Bulyan, which was too computationally expensive for VGG11 setting considered in the main submission. We also provide the results with Median for completion.

In our experiments, we use the Pytorch (Paszke et al., 2019) library for implementing the functionalities associated with training.

A. Prior Works

We recall from Section 2 that for iteration $r \in \{1,\dots,r_{\max}\} = [r_{\max}]$, g_j^r denotes the stochastic gradient computed by client $j \in \{1,\dots,n\}$. In the following, we describe the important prior algorithms that we use as benchmarks for Byzantine-resilient federated learning in our experiments.

Median (Yin et al., 2018): The master computes the coordinate-wise median of the messages received from the clients. Specifically, for iteration r, the master receives the messages $\{z_j^r\}_{j=1}^n$, where $z_j^r = g_j^r$ if $j \in \mathcal{N}^r$ and arbitrary if $j \in \mathcal{B}^r$. The master then computes the vector g^r=median$\{z_1^r,\dots,z_n^r\}$, where the u-th coordinate of g^r is the one dimensional median of $\{z_j^r(u)\}_{j=1}^n$, for $u\in[d]$. The master then takes the model update step as follows:

$$\theta^r = \theta^{r-1} - \mu^r \left(g^r + \lambda\theta^{r-1}\right). \tag{12}$$

Noisy Median (abbr. N-Median) (Chen et al., 2019): For iteration r, if client $j\in\mathcal{N}^r$, it uploads $z_j^r=g_j^r+\xi_j^r$ to the master, where $\xi_j^r\in\mathbb{R}^d$ and the entries $\{\xi_j^r(u)\}_{u=1}^d$ are drawn IID from a Gaussian distribution mean 0 and standard deviation σ_{N}. At the master, the median step and the parameter update step are carried out as described previously in the Median algorithm.

RSA (Li et al., 2019): The main idea in RSA is to reformulate the optimization problem in (2) as a consensus based optimization problem, and then incorporate a regularization term within the objective function to mitigate Byzantine faults. The proposed solutions are based on subgradients. Particularly, when the incorporated regularization term is based on ℓ_1-norm penalty, (Li et al., 2019) proposes the following set of updates:

$$\theta_j^r = \theta_j^{r-1} - \mu^r\Big(\frac{1}{n}g_j^r + \delta\text{sign}\left(\theta_j^{r-1}-\theta_M^{r-1}\right)\Big), \tag{13}$$

$$\theta_M^r = \theta_M^{r-1} - \mu^r\Big(\lambda\theta_M^{r-1} + \delta\sum_{j=1}^n \text{sign}\left(\theta_M^{r-1}-\theta_j^{r-1}\right)\Big), \tag{14}$$

where δ is a regularization parameter, and sign$(\cdot)$ is the coordinate-wise standard signum function. Therefore, in RSA, each client as well as the master maintains a copy of the model parameter. In iteration r, the master transmits θ_M^{r-1} to all the clients, each client $j\in\mathcal{N}^r$ computes the gradient of the empirical loss over a random data sample using its local model copy θ_j^{r-1}, and updates its copy using (13). Similarly, the master updates its copy of the model using (14). At the end of the iteration, each client $j\in\mathcal{N}^r$ uploads θ_j^r to the master, while client $j\in\mathcal{B}^r$ can upload an arbitrary value.

Krum (Blanchard et al., 2017): The main idea is to find the client whose gradient update is closest to the gradient updates of the nearest $n-f-2$ clients. Particularly, for client $i, j \in \{1,\dots,n\}$, let $dist(i,j) = \|z_i^r - z_j^r\|_2^2$, and let $dist(i)$ denote the sum of distances of z_i^r from the gradient updates of the nearest other $n-f-2$ clients. Then Krum selects the client i that has the minimum $dist(i)$, i.e.

$$i = Krum(\{z_j^r\}_{j=1}^n) = \arg\min_{j\in[n]} dist(j). \tag{15}$$

The master then updates the model using z_j^r as follows:

$$\theta^r = \theta^{r-1} - \mu^r\left(z_j^r + \lambda\theta^{r-1}\right). \tag{16}$$

Trimmed Mean (Yin et al., 2018; Xie et al., 2018; Guerraoui et al., 2018): In this family of Byzantine-resilient algorithms, the main idea is to trim large values for each dimension across client gradient updates, before aggregating the client updates. Depending on the algorithm, for the coordinate $u\in[d]$, let V_u denote subset of elements in $\{z_j^r(u)\}_{j=1}^n$ that will be aggregated. Then, the coordinate $u\in[d]$ of the trimmed mean gradient g^r is $g^r(u)=\frac{1}{|V_u|}\sum_{v\in V_u} v$. In (Yin et al., 2018), V_u is obtained by removing the largest βn and the smallest βn elements of $\{z_j^r(u)\}_{j=1}^n$, for the coordinate

$u \in [d]$, where $\beta \in [0, 1/2)$ is a hyperparameter. The authors in (Xie et al., 2018) propose to construct V_u by selecting only the top-$(n-f)$ values in $\{z_j^r(u)\}_{j=1}^n$ that are closest to the median of $\{z_j^r(u)\}_{j=1}^n$. Finally, in (Guerraoui et al., 2018), the authors propose to construct V_u by selecting only the top-$(n-2f)$ values in $\{z_j^r(u)\}_{j=1}^n$ that are closest to the median of $\{z_j^r(u)\}_{j=1}^n$. The master then updates the model using g^r as follows:

$$\theta^r = \theta^{r-1} - \mu^r \left(g^r + \lambda \theta^{r-1}\right). \tag{17}$$

Bulyan (Guerraoui et al., 2018): This involves a recursive application of Krum to find a 'selection set' of size $(n-2f)$ of possibly benign clients. Specifically, in the first step, Krum is applied on $\{z_j^r\}_{j=1}^n$ to select the client i that has minimum sum of distances to $n-f-2$ clients, and adds z_i^r to the selection set. Then, Krum is applied on the remaining set of updates $\{z_j^r\}_{j=1, j\neq i}^n$. This is repeated until the size of selection set is $n' = (n-2f)$. Thereafter, the master aggregates the gradient updates in the selection set using the trimmed mean algorithm proposed in (Guerraoui et al., 2018) and described above, wherein for each dimension, only the top-$(n'-2f)$ values closest to the median is considered for aggregation.

B. Complete Experimental Results for Section 4

We provide all the results for non-targeted attacks considered in the experimental settings in Section 4. We recall that in Section 4, we included the results for no attack, Gaussian attack and sign flip attack, while excluded the results for same value attack and label flip attack. We now provide all the results for the experimental settings in Section 4, including the ones provided in the main submission for quick reference.

B.1. Softmax Regression

In Fig. 9, results for softmax regression with MNIST dataset are provided. DiverseFL performs significantly well, almost matching the performance of OracleSGD in all scenarios.

B.2. Neural Networks

Neural Network Training with MNIST: In Fig. 10, results for neural network training with MNIST dataset are illustrated. We observe that DiverseFL performs consistently well in all scenarios, significantly outperforming other Byzantine-resilient approaches. Furthermore, increasing the data sharing percentage per client from 1% to 3% does not provide any noticeable advantage in convergence performance.

Neural Network Training with CIFAR10/CIFAR100: In Table 1, we provide the details of the VGG11 neural network architecture used for CIFAR10. Each convolutional layer is followed by group norm (Wu & He, 2018), ReLU, dropout (with dropout probability 0.2) and max pool (with kernel size 2×2 and stride 2). For group norm, we set each group parameter such that each group has 16 channels throughout the network. Before the first fully connected layer, we have an average pool (with output size 1×1). Each of the first two fully connected layers is followed by ReLU and dropout (with dropout probability 0.5). All biases are initialized to 0, while weights of group norm layers are all initialized to 1. Glorot uniform initializer is used for weights in convolutional layers, while default Pytorch initialization is used for weights in fully connected layers. For CIFAR100, the neural network architecture details are same as in Table 1, except the last fully connected layer, which has an output of size 100.

We illustrate the results for neural network training with CIFAR10 and CIFAR100 datasets in Fig. 11 and Fig. 12 respectively. We observe that for both CIFAR10 and CIFAR100, the performance of N-Median is significantly low due to increase in mini-batch sampling noise, overpowering the benefits of adding the Gaussian noise by the benign clients. DiverseFL performs quite well for both of the complex datasets. Furthermore, for CIFAR100, the performance of DiverseFL increases slightly when the sample sharing percentage is increased from 1% to 3%.

C. Additional Experiments

We now provide the additional experimental results for non-targeted attacks for CIFAR10 for a smaller neural network, considering both Bulyan and Median as well. In Table 2, we provide the details of the smaller neural network architecture used for CIFAR10. First convolutional layer is followed by ReLU and maxpool with kernel size 3×3 and stride 3, while the second one is followed by ReLU and maxpool with kernel size 4×4 and stride 4. Each of the first two fully connected layers is followed by ReLU. All biases are initialized to 0. Glorot uniform initializer is used for weights in convolutional layers, while default Pytorch initialization is used for weights in fully connected layers. The results for this setting are illustrated in Fig. 13. We can observe from the results that DiverseFL almost matches the performance of OracleSGD in all scenarios. Furthermore, for the same value attack, all except DiverseFL and OracleSGD perform quite poorly. While N-Median performs better than Median and Bulyan in other scenarios, its performance is quite low in comparison to DiverseFL. Finally, the performance of DiverseFL improves in the presence of label flip attack when per client data sharing percentage is increased from 1% to 3%, it remains almost same for the other scenarios.

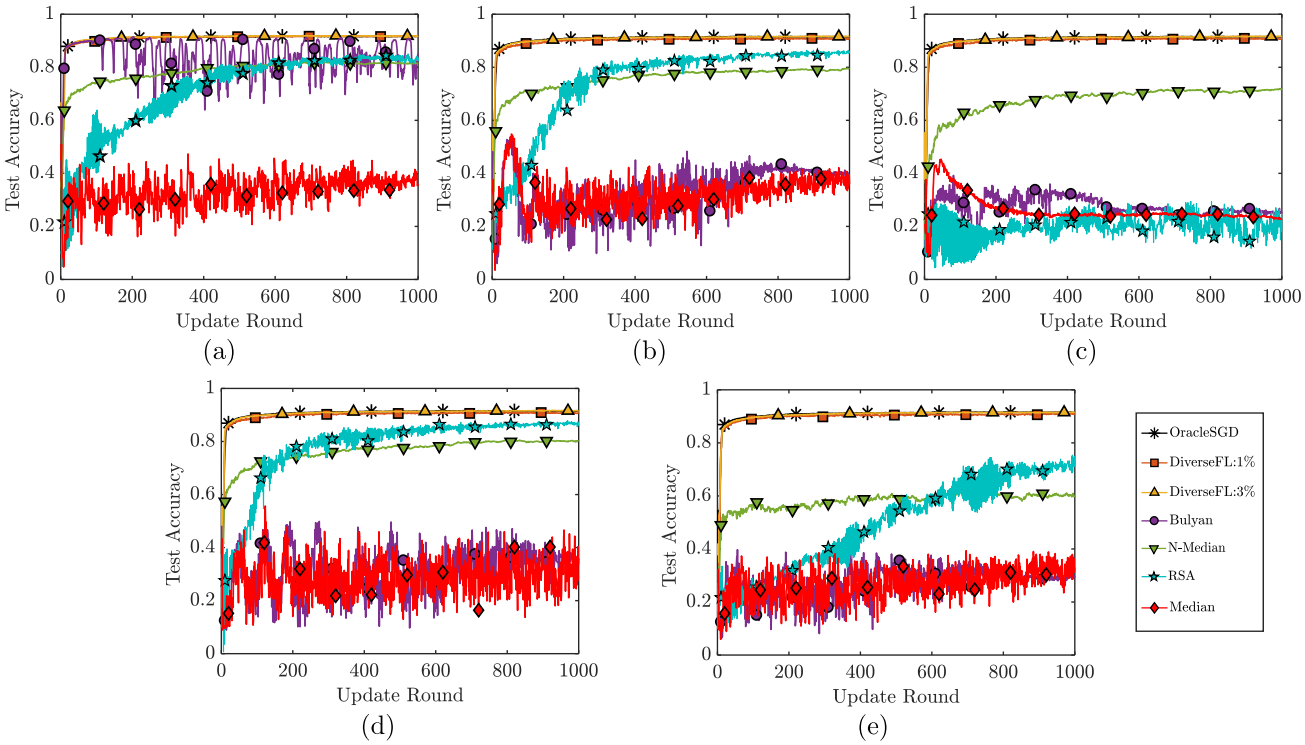

Figure 9. Top-1 accuracy for the softmax regression setting with MNIST in Section 4.1: (a) No Attack; (b) Gaussian Attack; (c) Sign Flip Attack; (d) Same Value Attack; (e) Label Flip Attack.

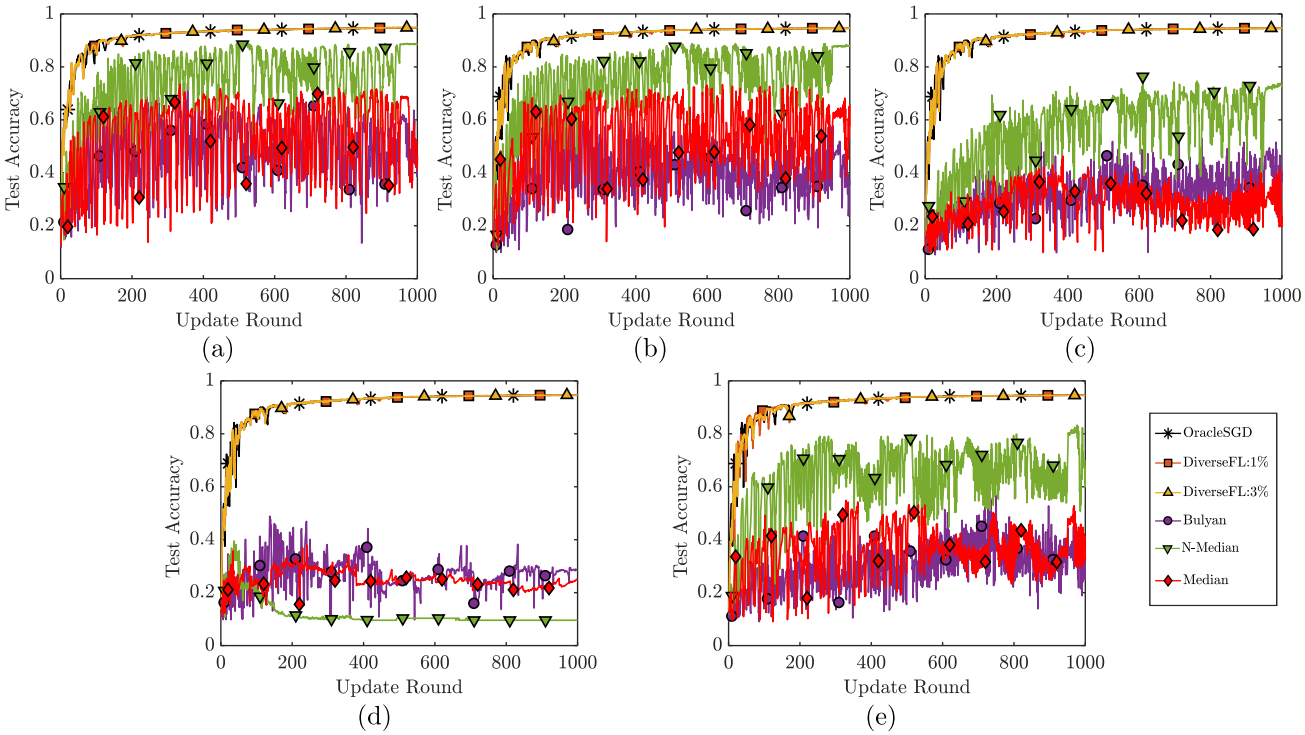

Figure 10. Top-1 accuracy for the neural network training setting with MNIST in Section 4.2: (a) No Attack; (b) Gaussian Attack; (c) Sign Flip Attack; (d) Same Value Attack; (e) Label Flip Attack.

Table 1. Detailed information of the architecture of the VGG11 network used in our CIFAR10 experiments.

Sl. No.	Parameter	Shape	Hyperparameters
1	Conv2d	$3 \times 64 \times 3 \times 3$	stride$= 1$, padding$= (1, 1)$
2	Conv2d	$64 \times 128 \times 3 \times 3$	stride$= 1$, padding$= (1, 1)$
3	Conv2d	$128 \times 256 \times 3 \times 3$	stride$= 1$, padding$= (1, 1)$
4	Conv2d	$256 \times 256 \times 3 \times 3$	stride$= 1$, padding$= (1, 1)$
5	Conv2d	$256 \times 512 \times 3 \times 3$	stride$= 1$, padding$= (1, 1)$
6	Conv2d	$512 \times 512 \times 3 \times 3$	stride$= 1$, padding$= (1, 1)$
7	Conv2d	$512 \times 512 \times 3 \times 3$	stride$= 1$, padding$= (1, 1)$
8	Conv2d	$512 \times 512 \times 3 \times 3$	stride$= 1$, padding$= (1, 1)$
9	Linear	512×4096	-
10	Linear	4096×4096	-
11	Linear	4096×10	-

Table 2. Detailed information of the architecture of the smaller network used in our CIFAR10 experiments in Section 8.

Sl. No.	Parameter	Shape	Hyperparameters
1	Conv2d	$3 \times 16 \times 3 \times 3$	stride$= 1$, padding$= (1, 1)$
2	Conv2d	$16 \times 64 \times 4 \times 4$	stride$= 1$, padding$= (0, 0)$
3	Linear	64×384	-
4	Linear	384×192	-
5	Linear	192×10	-

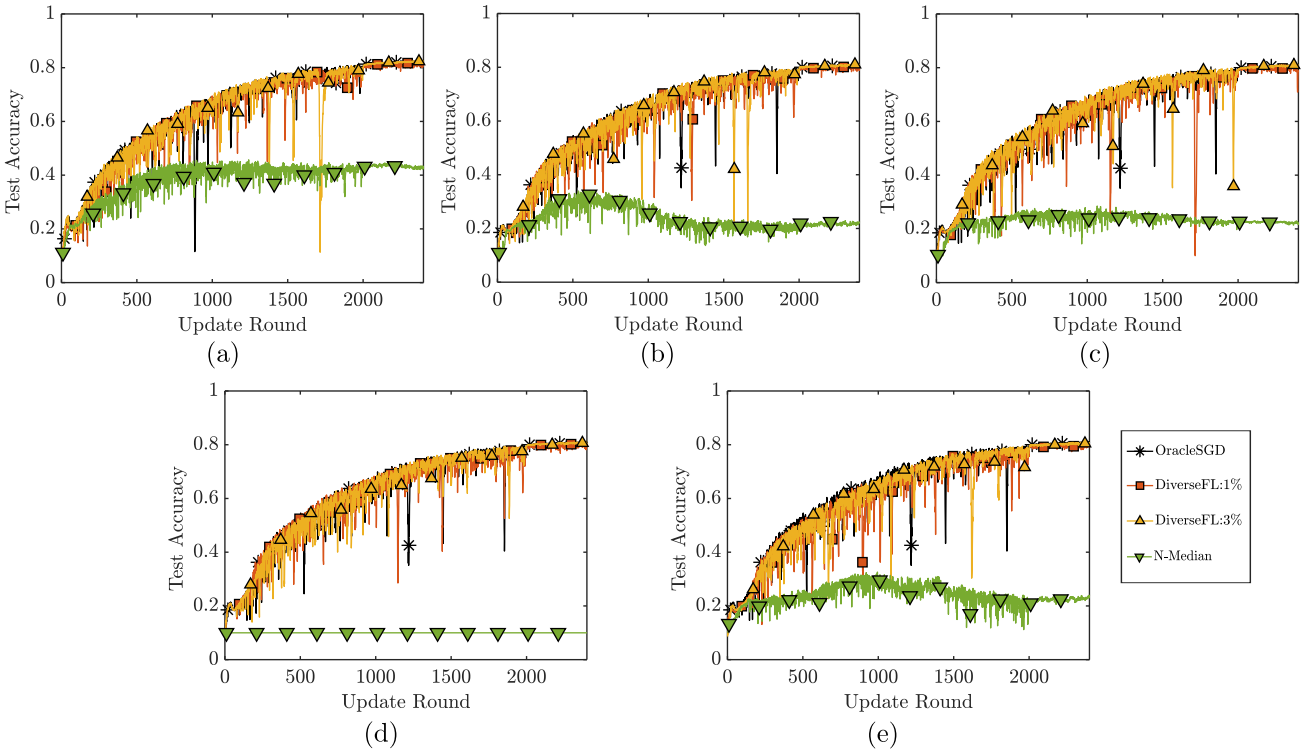

Figure 11. Top-1 accuracy for the neural network training setting with CIFAR10 in Section 4.2: (a) No Attack; (b) Gaussian Attack; (c) Sign Flip Attack; (d) Same Value Attack; (e) Label Flip Attack.

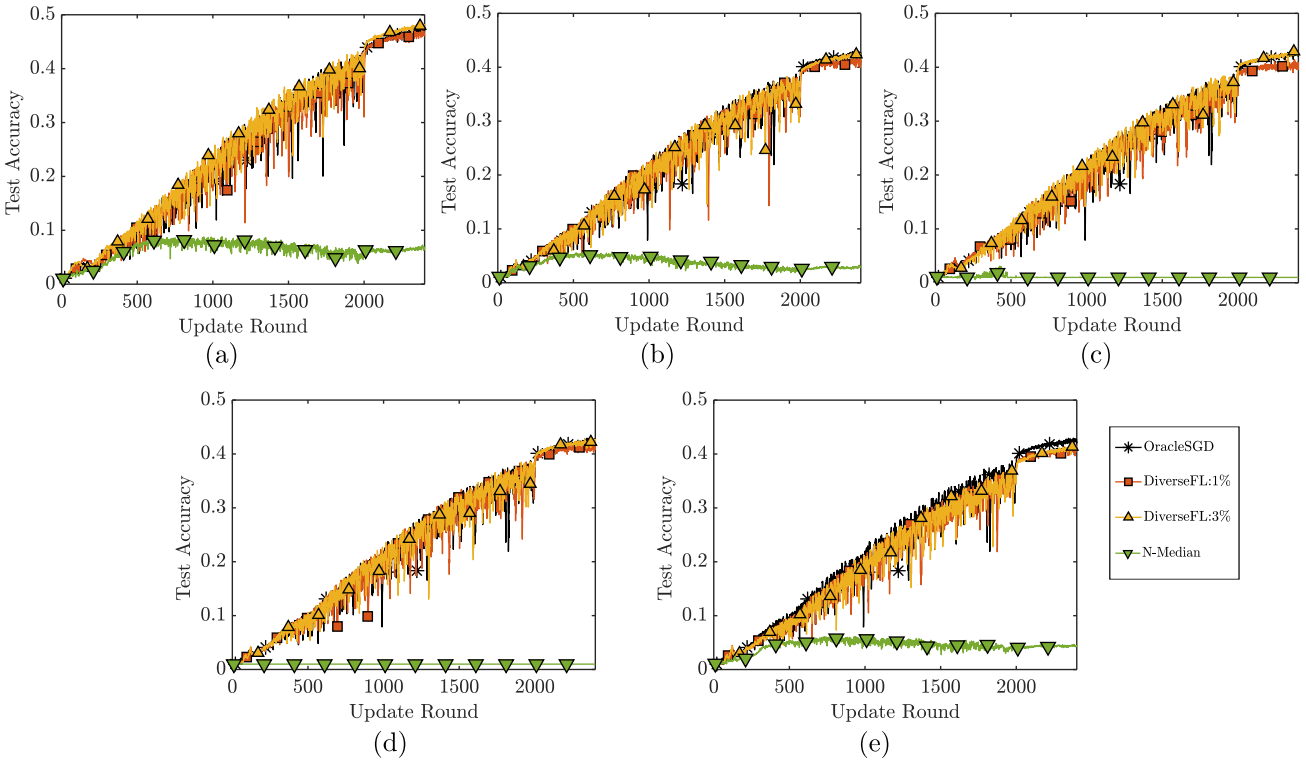

Figure 12. Top-1 accuracy for the neural network training setting with CIFAR100 in Section 4.2: (a) No Attack; (b) Gaussian Attack; (c) Sign Flip Attack; (d) Same Value Attack; (e) Label Flip Attack.

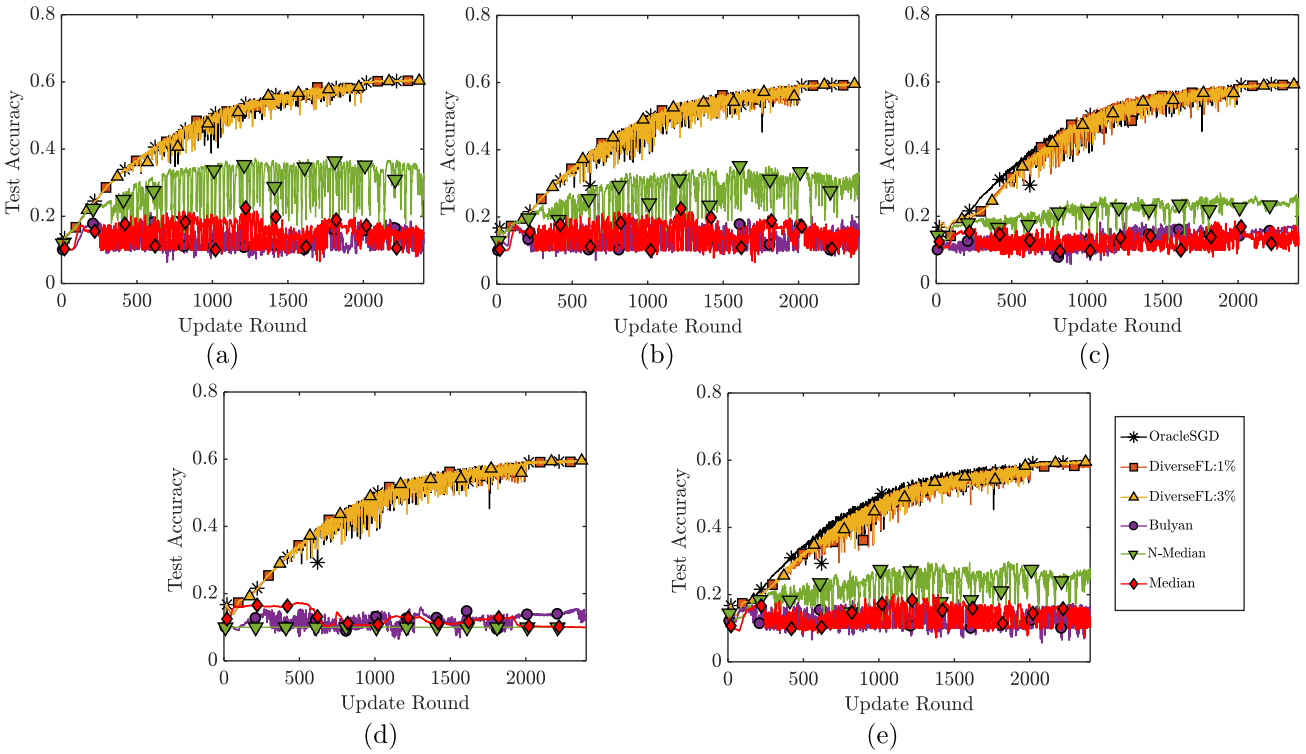

Figure 13. Top-1 accuracy results for the additional experiments with a smaller neural network with CIFAR10: (a) No Attack; (b) Gaussian Attack; (c) Sign Flip Attack; (d) Same Value Attack; (e) Label Flip Attack.