SERVER AGGREGATION AS LINEAR REGRESSION: REFORMULATION FOR FEDERATED LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

We propose a conceptually novel framework for Federated Learning (FL) called FedFit to mitigate issues of FL. FedFit is a reformulation of the server aggregation in FL, where the global model is updated by *linear regression*. This reformulation naturally enables us to utilize the established linear regression techniques for several FL issues. For example, we apply robust regression to alleviate the vulnerability issue against attacks on the global model from collapsed clients, and we apply LASSO regression to introduce sparsity into the model to reduce the communication cost in FL. Moreover, FedFit enables clients to upload compressed model parameters to the server, significantly reducing the data traffic. In experiments, we demonstrate that FedFit successfully improves robustness against attacks on a global model by robust regression and reduces the global model size by LASSO regression.

1 INTRODUCTION

On the growing demand for using massive user data for machine learning while ensuring data privacy, Federated Learning (FL) has been proposed by (McMahan et al., 2017; Mohri et al., 2019). In FL, a machine learning model is first trained on each distributed data, i.e., each client's data, and then a server aggregates a trained model received from each client to make one global model. Contrary to a traditional machine learning philosophy, the server cannot access each client's data in the training pipeline of FL. Instead, the client's trained models are only accessible from the server. Users do not need to upload their own data to the server, so FL can protect user privacy.

Recent studies show that FL has several challenges in practical scenarios (Li et al., 2020b). The first challenge is robustness against attacks from malicious clients (Bagdasaryan et al., 2020; Bhagoji et al., 2019). In FL, a global model is updated by aggregating clients' trained models. When malicious clients are connected to the server, such clients can upload an adversarial model to change the global model's behavior intentionally. The second challenge is to reduce communication costs (Konečný et al., 2016; Singhal et al., 2021). Since modern deep neural network models contain many parameters (Dosovitskiy et al., 2020; Tolstikhin et al., 2021), the server's and client's communications consume much energy.

To tackle these problems, we focus on the fact that similar problems have been discussed in the field of linear regression, such as protecting a model from outliers and making the model sparse. To take advantage of existing techniques on linear regression, we consider the relationship between FL and linear regression and *reformulate the server aggregation of FL as the linear regression*. We name our method **Fed**erated Learning as **Fit**ting problems (**FedFit**). The reformulation enables us to utilize the established regression techniques as solutions for the challenges in FL. For example, robust linear regression can handle outliers, which could be applicable for protecting the global model from collapsed client attacks. In addition, LASSO regression introduces the sparsity to the weights, so this technique can reduce the communication costs of the model between the server and a client. Furthermore, FedFit significantly reduces data traffic since clients upload a compressed model to the server.

In summary, our contributions are as follows:

- We propose FedFit, which reformulates the model aggregation of FL as the linear regression. It enables FL to use the benefit of linear regression techniques.

- In FedFit, a model is compressed when clients send back a model to the server, which reduces data traffic to upload a model to the server.
- We utilize linear regression techniques as a solution for FL problems. We experimentally show that the robust regression successfully improves the robustness against attacks from collapsed clients, and the reduction of global model size is achieved by LASSO regression.

2 Related Work

The recent spread of edge devices and massively distributed data in each device for machine learning has motivated researchers to propose Federated Learning (FL) (McMahan et al., 2017). As an application of FL, some methods are proposed across various research fields, such as computer vision, natural language processing, and speech processing (Leroy et al., 2019).

One of the challenges in FL is the data distribution shift among clients. In some practical scenarios of FL, each client's data has a different distribution, and (Fallah et al., 2020; Zhang et al., 2021) proposed methods to create a personal model suitable for each own local data, while (Diao et al., 2021; Mohri et al., 2019) proposed methods to train the global model under heterogeneous data. Although FL is initially proposed to protect clients' data privacy, this scheme is valunerable to attacks from malicious clients at the same time (Bagdasaryan et al., 2020; Fung et al., 2018). Therefore, several studies proposed methods to remove such malicious data (Tuor et al., 2020) or models (Cao et al., 2021) by focusing on their distributions.

Another challenge of FL is that FL requires frequent communications of models between clients and a server, leading to other problems such as huge communication costs (Konečný et al., 2016; Li et al., 2020a). For this challenge, some studies compress a model to decrease the communication costs of the model between a server and clients (Reisizadeh et al., 2020; Singhal et al., 2021) or introduce scheduling strategies (Yang et al., 2019) to communicate a model between clients and a server efficiently.

Position of this paper: FedFit is an aggregation method. Specifically, FedFit reformulates it as linear regression. Thanks to this, we utilize the established linear regression method to solve FL limitations through our reformulation.

3 Background of Federated Learning

There are two phases in the process of FL: local training and server aggregation. At the first stage of FL, the server distributes a global model to each connected client. After receiving a global model, the client initializes a model with the weights of the global model and trains the model with its local data. Such a trained model is called the local model, and this training is called local training. Then, each client sends a local model to the server when local training is finished. The server collects these local models and uses them to update one global model. This update process of the global model is called server aggregation. There are several aggregation methods (Bonawitz et al., 2016; Ek et al., 2021), and one standard aggregation method is the Federated Average (FedAvg) (McMahan et al., 2017). FedAvg updates the global model by simple averaging of collected local models as follows:

$$\mathbf{w_g} = \frac{1}{n}\sum_{i}^{n} \mathbf{w_l}^{i}. \tag{1}$$

Here $\mathbf{w_g}$ is a global model, and n is the number of clients, i indicates i-th client, and $\mathbf{w_l}^i$ is a local model trained using i-th client's data. The consecutive process of the local training and the server aggregation is called a communication round. FL repeatedly executes this communication round until the global model is converged.

4 Our Method

The main objective of FedFit is to formulate server aggregation as linear regression. In this section, we first review linear regression in Sec. 4.1, then present the pipeline of FedFit in Sec. 4.2.

Table 1: Challenges of FL and corresponded applicable linear regression techniques to such challenges.

Challenges of FL (Li et al., 2020b)	Corresponded applicable techniques
Robustness against attacks from collapsed clients	Robust regression (see Sec. 5.2)
Reduction of communication cost	LASSO regression (see Sec. 5.3)

4.1 Recap of Linear Regression

Linear regression is the fitting problem to approximate the function that represents the linear relationships between input and output pairs. Here, let us consider the 1d output linear model. Given input vector $\mathbf{x} \in \mathbb{R}^d$ and target vector $t \in \mathbb{R}$, the linear regression is problem to obtain optimal function $f_{\mathbf{w}}(\mathbf{x})$ by learnable weights $\mathbf{w} \in \mathbb{R}^d$ to satisfy:

$$\begin{aligned} t &= f_{\mathbf{w}}(\mathbf{x}), && (2)\\ &= w_1x_1 + ... + w_dx_d, && (3)\\ &= \mathbf{w}^T\mathbf{x}. && (4) \end{aligned}$$

There are several linear regressions with various loss functions for different purposes, and one of the standard methods is the least-square method. The optimal weights $\mathbf{w}^* \in \mathbb{R}^d$ are obtained by minimizing a square loss L between the target vector t and output vector of a linear model $f_{\mathbf{w}}(\mathbf{x})$:

$$\begin{aligned} L(\mathbf{w}; t, \mathbf{x}) &= \frac{1}{2}\sum_{i=1}^{n}(f_{\mathbf{w}}(\mathbf{x}_i) - t_i)^2, && (5)\\ \mathbf{w}^* &= \underset{\mathbf{w}}{\operatorname{argmin}}\, L(\mathbf{w}; t, \mathbf{x}), && (6) \end{aligned}$$

where n represents the number of input and target vector pairs, and i indicates i-th data of n.

Although this example is one basic case, some research on linear regression has extended this problem under different settings and proposed multiple solvers for each different setting. Rethinking this knowledge, we are motivated to utilize these solvers to solve the challenges of FL. Table 1 summarizes examples of applications of linear regression solvers to FL challenges. As solvers for those challenges, we consider robust and LASSO regression are promising. In addition to those techniques, many extensions could be considered. We focus on applying these two techniques in this paper, and we leave other applications of regression techniques for FL settings to future work.

4.2 FedFit

4.2.1 Overview

FedFit is an aggregation method that consists of two processes: weights compression in clients and weights reconstruction through linear regression in the server. Figure 1 visualizes the overview of FedFit. In usual aggregation, such as FedAvg, clients upload the trained local model, and the server aggregates the collected local model by simply averaging received weights. Instead of averaging aggregation, FedFit asks clients to upload compressed weights of the trained local model (detailed in Sec. 4.2.2), where shared key vectors are used for compression. The server then updates the global model using uploaded compressed weight and shared key vectors through some solvers of linear regression (detailed in Sec. 4.2.3).

In the rest of this section, let us consider one fully-connected layer without bias for simplicity. Assume the input dimension is d and the output dimension is o. Here, the weights are denoted as $\mathbf{w} \in \mathbb{R}^{o \times d}$. We illustrate each stage of FedFit in detail.

4.2.2 Weight compression by matrix product in client

In order to achieve linear regression on the server, we design a compression-based model uploading process for clients. Local model weights are compressed by matrix product after finishing local training. To calculate the matrix product, we regard model weights as 2D vectors to calculate the matrix product. Additionally, a client knows some key vector $\mathbf{v}$, which is shared between the server, to calculate the matrix product. The dimension of the key vector is determined based on the dimensions

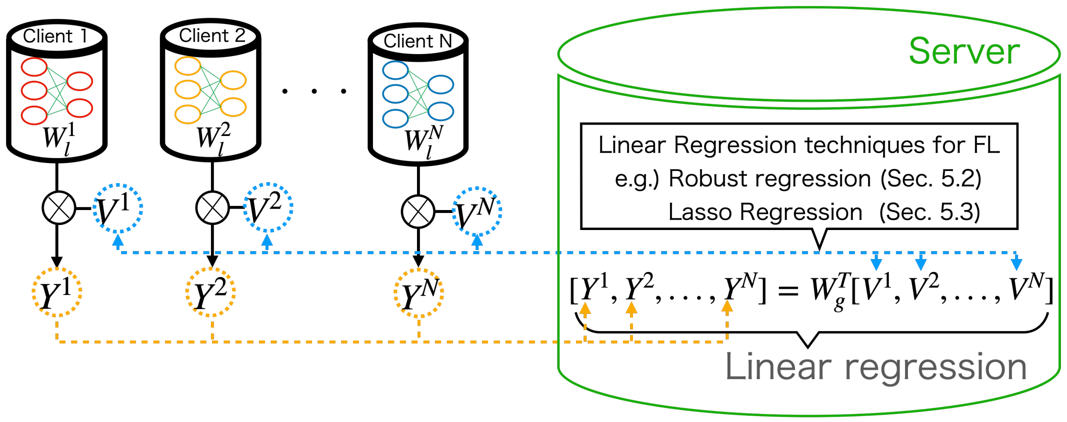

Client 1
Client 2
Client N
W_l^1
W_l^2
W_l^N
V^1
V^2
V^N
Y^1
Y^2
Y^N
Server
Linear Regression techniques for FL
e.g.) Robust regression (Sec. 5.2)
Lasso Regression (Sec. 5.3)
$[Y^1, Y^2, \dots, Y^N] = W_g^T [V^1, V^2, \dots, V^N]$
Linear regression

Figure 1: Overview of FedFit. The algorithm of FedFit is constructed as follows: (1) Each client trains a local model $\mathbf{w_l}$ by local data. (2) Each client compresses local model $\mathbf{w_l}$ by using a key vector $\mathbf{v}$, and uploads compressed local model $\mathbf{y}$. (3) Server reconstructs the global model $\mathbf{w_g}$ by using linear regression techniques. Here, the same key vector is shared between the server and each client. $\otimes$ represents the matrix product.

of model weights, and the key vector $\mathbf{v}$ has $\mathbb{R}^{o \times k}$ dimensions, where k controls the compression degree. We compress model weights by using key vectors $\mathbf{v}$ such as $\mathbf{y} = \mathbf{w}^T \mathbf{v} \in \mathbb{R}^{d \times k}$. Then, each client uploads such compressed weights $\mathbf{y}$ to the server.

4.2.3 Server aggregation as linear regression

By carefully collecting compressed weights and shared key vectors, server aggregation can be reformulated as linear regression. Let us consider $\mathcal{N}$ clients are connected in each communication round, and for simplicity of explanation, let us consider the dimension of key vector k as 1. The server can collect $\mathcal{N}$ compressed weights $\mathbf{y}$ from clients and can know corresponding key vectors $\mathbf{v}$ which were used during model compression. Therefore, the server has $\mathbf{Y} = [\mathbf{y}^1, ..., \mathbf{y}^i, ..., \mathbf{y}^{\mathcal{N}}]$ for compressed weights and $\mathbf{V} = [\mathbf{v}^1, ..., \mathbf{v}^i, ..., \mathbf{v}^{\mathcal{N}}]$ for key vectors, where i indicates i-th client of $\mathcal{N}$ clients. Here, let us look back at the process of weight compression in each client. If only one paired data, e.g., $\mathbf{y}^i$ and $\mathbf{v}^i$, is obtained in the server, the server needs to decode weights $\tilde{\mathbf{w}}$ that satisfies $\mathbf{y}^i = \tilde{\mathbf{w}}^T \mathbf{v}^i$. Note that the server has $\mathcal{N}$ client data in practice, and the optimal *one* global model should be reconstructed to satisfy this decoding process for all client data as much as possible. Therefore, in the server aggregation, the global model $\tilde{\mathbf{W}}_\mathbf{g} \in \mathbb{R}^{o \times d}$ should be reconstructed to satisfy:

$$\mathbf{Y} = \tilde{\mathbf{W}}_\mathbf{g}^T \mathbf{V}, \tag{7}$$

where $\mathbf{Y} \in \mathbb{R}^{d \times \mathcal{N}}$ and $\mathbf{V} \in \mathbb{R}^{o \times \mathcal{N}}$. Thanks to this reformulation, we can apply solvers of linear regression to calculate the global model weights $\tilde{\mathbf{W}}_\mathbf{g}$ because the objectives of Eq. (7) is the equivalent to the linear regression by considering $\mathbf{Y}$ *as target vectors and* $\mathbf{V}$ *as input vectors*, as shown in Eq. (4). The strength of this reformulation is that we are able to choose the linear regression solver depending on the problem of FL. As can be seen in Table 1, robust regression can be applied when collapsed clients upload an adversarial model $\mathbf{y}_i$, and we can obtain a sparse global model by LASSO regression to reduce communication costs.

To solve Eq. (7), we use gradient descent by setting initial parameters as pre-updated global model parameters since gradient descent could be applicable to update parameters regardless of chosen loss functions. This is written as follows.

$$\begin{aligned} W_0 &= W_g, && (8)\\ W_t &= GradientDescent(W_{t-1}, L(\tilde{W}; Y, V)), && (9) \end{aligned}$$

Algorithm 1 Compression via the matrix product after local training of i-th client. The $\mathcal{L}$ layers are indexed by l. θ_l represents l-th layer of trained local model weights.

```
for All layers do
    Get a key vector $\mathbf{v}_l$
    Calculate the matrix product $\mathbf{y}_l = \theta_l^T \mathbf{v}_l$ for compression
end for
Concatenate compressed weights $\mathbf{y}^i = [\mathbf{y}_1, ..., \mathbf{y}_\mathcal{L}]$
return $\mathbf{y}^i$
```

Algorithm 2 Server aggregation as linear regression. $\mathcal{T}$ is the number of gradient descent in the server. $\mathbf{Wg}$ is the global model before updating.

```
for All layers do
    $\tilde{\mathbf{W}}_l^0$ = $\mathbf{Wg}_l$
    Collect $\mathbf{Y}_l$ and $\mathbf{V}_l$
    for t = 1, 2, ..., $\mathcal{T}$ do
        $\tilde{\mathbf{W}}_l^t$ = $GradientDescent(W_{t-1}, L(\tilde{W}; Y, V))$ by Eq. (9)
    end for
    $\tilde{\mathbf{W}}\mathbf{g}_l = \tilde{\mathbf{W}}_l^\mathcal{T}$
end for
```

where L is the loss function depending on the problem settings. For example, $L = ||\mathbf{Y} - \tilde{\mathbf{W}}_\mathbf{g}^T \mathbf{V}||_2^2$ is a naive loss function to reconstruct weights, and Eq. (11) can be used for L to ensure the robustness of a global model against attacks from collapsed clients. Alternatively, Eq. (12) can be used for L to sparsify the global model to reduce communication costs. For GradientDescent, SGD or Adam (Kingma & Ba, 2015) can be used to minimize a loss function L such as $W_t = W_{t-1} - \gamma \frac{\partial L(\tilde{W}; Y, V)}{\partial \tilde{W}}$, where γ is a learning rate. In this section, we demonstrated the reformulation for one fully-connected layer, and the reformulation can be applied similarly if there are multiple layers.

Although the above example assumes a fully-connected layer, modern deep models include other layers such as a convolution layer. The convolution layer's weights $\mathrm{C_w} \in \mathbb{R}^{n_o \times n_i \times k_h \times k_w}$ have more axes of dimensions compared to the fully-connected layer, where n_o is the output channel, n_i is the input channel, k_h and k_w are the height and width of the kernel, respectively. We reshape the dimensions of convolutional weights such as $\hat{\mathrm{C}}_\mathrm{w} \in \mathbb{R}^{n_o \times n_i k_h k_w}$. Then, the convolution layers are treated similarly as the fully-connected layer.

Finally, we present the overall algorithms regarding weight compression *after* local training and server aggregation as linear regression in Algorithm 1 and 2, respectively.

4.3 Discussion of Key Vectors

Sharing. There are multiple options for how sharing key vectors between a client and the server. One option is to generate key vectors in a client/server, then send them to another. As an efficient alternative method, if a client and server mutually agree on the generation rule about key vectors in advance, sharing only a seed to generate key vectors might be possible. Therefore, the additional communication cost of sharing key vectors is negligible.

Types. We generate key vectors $\mathbf{v}$ following gaussian distribution in this paper. This indicates that model compression is the same as random projection (Dasgupta & Gupta, 2003), which is known as an effective dimension reduction method (Bingham & Mannila, 2001) and is also known to be applicable for linear regression (Maillard & Munos, 2012). Although we experimentally show our proposed strategy using randomly generated key vectors, other key generation methods may be applicable, so we leave it as a research question in future work.

5 Experiments

5.1 Setup

In this section, we conduct two experiments to demonstrate that linear regression techniques can be fit to solve FL challenges through FedFit reformulation by: (i) ensuring the robustness of the global model against attacks from collapsed clients and (ii) sparsifying the global model to reduce the communication cost. The experiments are conducted on Fashion-MNIST and CIFAR-10 for image classification and IMDB and AGNews for text classification tasks. We use CNN and LSTM for each task, respectively. For the federated scenario, we split the training dataset into 100 subsets and assign one of them to each client. A local model is trained with CrossEntropyLoss. As an optimizer, SGD is used in all tasks during local training, and to solve linear regression in server aggregation (Eq. (9), we use the Adam optimizer. We set iteration times $\mathcal{T}$ in the server to 100. As key vectors $\mathbf{v}$, we sampling vector following gaussian distribution such as $\mathbf{v} \sim N(0, \frac{1}{k})$, where k controls compression degree as described in Sec. 4.2.2. Unless expressly noted, we compress the kernels of the convolution layer and output channel of the other layer in experiments. For detailed dataset information, hyperparameters, and architecture of CNN and LSTM, see Appendix A.

5.2 Robustness against attacks to the global model

Motivation and implementation details. For the possibility of malicious clients' participation, we need to consider situations such that a client uses malicious data during local training and the such client tries to replace the global model with any local model $\widehat{\mathbf{w_l}}$ by calculating Eq. (10) (Bagdasaryan et al., 2020).

$$\widehat{\mathbf{w_l}} = s\mathbf{w_l} - (s-1)\mathbf{w_g}, \tag{10}$$

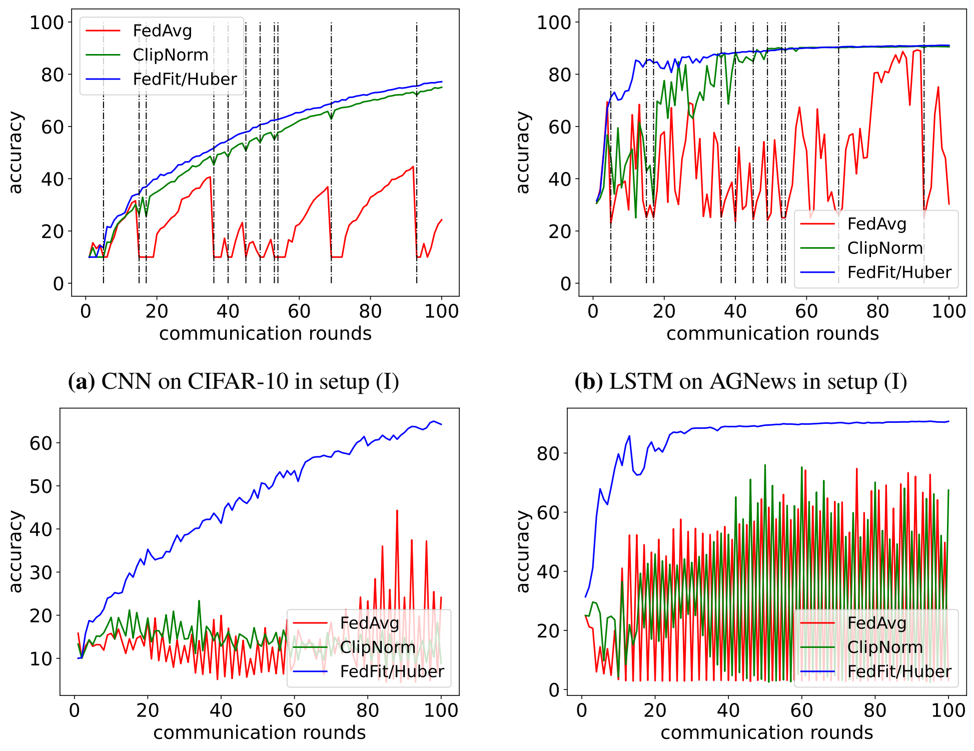
where s and $s-1$ are scale parameters and s is the same as the number of connected clients in an ideal attack. $\mathbf{w_g}$ is the global model received from the server. It is assumed that the poisoned models from collapsed clients greatly differ from the trained local models from non-collapsed clients. In other words, the parameter distribution of the poisoned model is regarded as outliers from the server's viewpoint compared with the local models that are not poisoned. In the field of linear regression, researchers have discussed how to handle outliers, and as a solution for that, Huber loss is one standard method. The Huber loss is defined as:

$$L_{Huber}(x,y) = \begin{cases} \dfrac{(x-y)^2}{2} & (|x-y| \leq \delta), \\ \delta(|x-y| - \frac{\delta}{2}) & (|x-y| > \delta), \end{cases} \tag{11}$$

where $\delta \in \mathbb{R}$ is a threshold value to reduce the effect of outliers. We can also apply L_{Huber} to FL to ensure robustness against attacks from collapsed clients through the reformulation of FedFit. The server selects 30 out of 100 clients in our experiments in one communication round. For attacking, we set up two scenarios: (I) one anonymous collapsed client participates FL communication with a 10% probability. (II) For more challenging settings, ten anonymous collapsed clients always participate FL communication and collaboratively attempt to degrade the global model performance. The collapsed client trains a local model with data whose labels are randomly permutated, and a trained local model is sent back to the server by following Eq. (10) for the attack. When collapsed clients collaboratively participate, the permutated label pattern is shared among collapsed clients, and the scale of Eq. (10) is modified accordingly.

In the experiments, we set δ in Eq. (11) to 0.001. We evaluate the performance of the global model by applying Huber loss in Eq. (9) for updating, and we verify robust regression is an effective method for defense through the reformulation of FedFit.

Results. Experimental results for robustness against attacks from collapsed clients are presented in Figure 2. For comparison of the defense method, we use clip norm defense (Sun et al., 2019). If the attacks by Eq. (10) succeed, the global model's performance is degraded in setup (I). As can be seen in Figure 2, FedAvg drops the performance when the collapsed client joins the communication round (i.e., vertical dot line). On the contrary, robust regression through FedFit and clip norm maintain its performance against the attack from the collapsed client. The setup (II) is more challenging, and the global model performance is degraded through communication rounds, yet FedFit still achieves



(a) CNN on CIFAR-10 in setup (I)

(b) LSTM on AGNews in setup (I)

(c) CNN on CIFAR-10 in setup (II)

(d) LSTM on AGNews in setup (II)

Figure 2: The performance of the global model against attacks from collapsed clients. 30 clients are selected in one communication round. Black vertical dot lines indicate the rounds when the malicious client is connected to the server in setup (I). Here, setup (I) is one collapsed client case, and setup (II) is ten collapsed clients case as described in motivation.

stable updates. Here collapsed clients attempt to replace the global model in every communication round in setup (II), so if the defense does not properly work, there seems to be some vibration since successive replacement cancels out the effects of attacking each other. As discussed in the motivation, robust regression is effectively applicable to FL through the reformulation of FedFit, and the global model is updated stably against attacks.

5.3 Sparse model to reduce communication cost

Motivation and implementation details. Although deep models contain many parameters, it is known that some of them are redundant and do not affect performance. We consider that a sparse model is a promising approach to reducing model size because the number of model parameters highly affects communication costs in FL. This issue motivates us to utilize LASSO regression in FL. By tackling server aggregation as the sparse regression, we can apply LASSO regression to obtain a sparse global model through FedFit. Formally, we use least-square loss with L1 penalty in Eq. (9) such as:

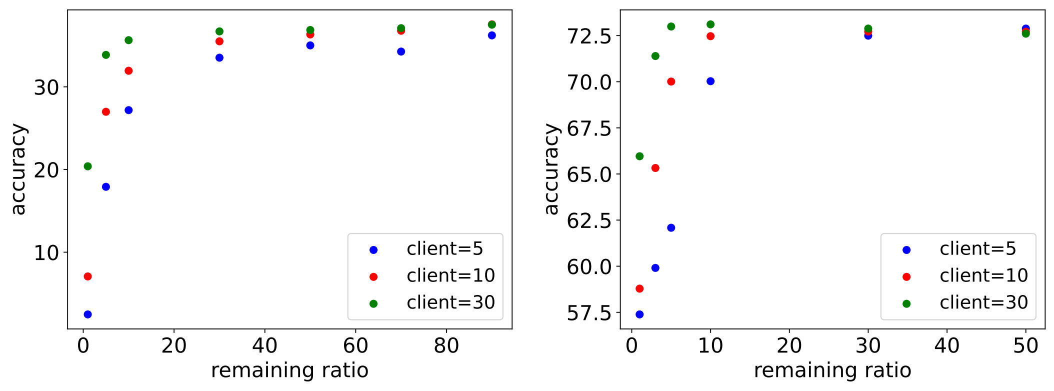
$$L = ||\mathbf{Y} - \tilde{\mathbf{W}}_{\mathbf{g}}^{T}\mathbf{V}||_2^2 + \lambda||\tilde{\mathbf{W}}_{\mathbf{g}}||_1. \tag{12}$$

Although some papers proposed methods for a sparse global model in FL (Parcollet et al., 2022), we aim to verify whether the LASSO regression approach works for sparsification. We thus evaluate the number of non-zero parameters of the global model by adding the L1 penalty to only local training, only server aggregation, or both in experiments for comparison. We regard the weights with smaller values than 1×10^{-6} as sparse weight. As an evaluation, we report the error rate and the number of the model's non-zero parameters after 100 communication rounds.

Results. We summarize the global model performance trained with the L1 penalty in Table 2. As shown in Table 2, FedAvg fails to reduce the number of the global model's non-zero parameters

Table 2: The result of the error rate and the number of non-zero parameters of the global model by the L1 penalty on the FashionMNIST (FMNIST) / IMDB dataset with CNN / LSTM, respectively. 30 clients are selected in each communication round. L1 (local / server) indicates that the L1 penalty is applied in local training and server aggregation, respectively. We set λ in Eq. (12) as 1×10^{-9} in FMNIST and 1×10^{-8} in IMDB.

method	L1 (local / server)	error $\downarrow$ (FMNIST / IMDB)	params [K] $\downarrow$ (FMNIST / IMDB)
FedAvg	no / no	9.02 / 27.12	4,505 / 572
FedAvg	yes / no	9.09 / 27.29	4,482 / 571
FedFit	no / no	9.55 / 26.89	4,505 / 572
FedFit	yes / no	9.52 / 27.07	4,481 / 571
FedFit	no / yes	9.87 / 27.91	**1,937 / 452**
FedFit	yes / yes	9.87 / 27.59	**1,937 / 451**



(a) CNN on CIFAR-100 **(b)** LSTM on IMDB

Figure 3: Performance evaluation by varying the compression degree during the local model uploading. We report the final global model performance after 100 communication rounds. Here client=x means x clients are selected in one communication round, and the horizontal axis means the remaining output channels ratio in each layer after model compression (aggressively compressed to the left).

by adding the L1 penalty during local training. We consider that the aggregation step in the server causes this low reduction. Since the sparse local models are different for each client, the simple averaging aggregation of FedAvg flattens such sparse weights and does not make a global model sparse. On the contrary, FedFit makes the global model sparse when adding the L1 penalty during server aggregation. In FedFit, server aggregation is now handled as linear regression. As shown in Table 2, FedFit successfully reduces the global model's non-zero parameters by using LASSO regression. As the strength of FedFit, we can obtain a sparse global model even if local models are trained at each client without design for sparsification.

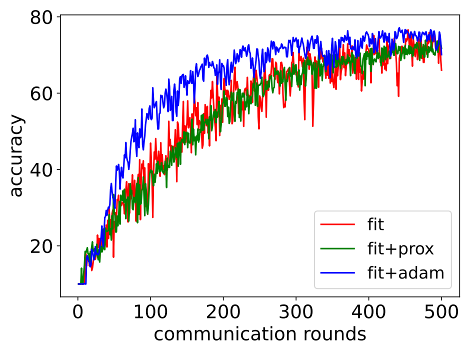
6 Analysis

To address the limitation and capability of FedFit, we conduct two experiments in this section. (i) We expect that the compression degree when uploading a compressed local model $\mathbf{y}$ from a client affects model performance. Therefore in the first experiments, we investigate the relationship between the uploaded model's compression ratio (controlled by key vectors) and model performance. (ii) FedFit is a new concept in server aggregation, and it is expected to be compatible with other FL techniques which change local training or other optimizations after aggregation in the server. Therefore in the second experiment, we combine FedFit with FedProx (Diao et al., 2021) and FedAdam (Reddi et al., 2021) and check model performance in the heterogeneous distribution case. For a detailed experimental setup, see Appendix A.4.

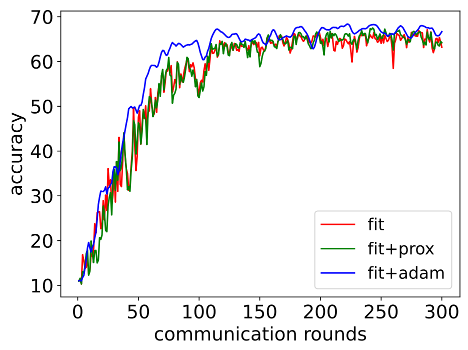
6.1 Tradeoff between model compression during uploading and performance

We evaluate the model performance by changing the compression degree during the uploading of a local model. Figure 3 shows the results of tradeoff experiments. Here, we compress output channels in all layers during model uploading in CNN. In the image classification task, we use CIFAR-100. As the figures show, compression degree during model uploading and participation of connected clients in one communication round affect the global model performance after aggregation. These results indicate that when the server has many contributed clients during communication rounds, the server allows high compression for uploading a model from each client to maintain the global model performance to reduce data traffic. Actual global model performance arose from such hyper-parameters depending on architecture or task, as the figure shown, so we need to decide on a suitable compression degree based on a task. It is an interesting research direction to dynamically determine the degree of compression considering bandwidth or the number of connected clients.

6.2 Combination with other FL methods in Heterogeneous Distribution



(a) CNN on CIFAR-10 in non-i.i.d. settings.



(b) LSTM on Yahoo Answers in non-i.i.d. settings.

Figure 4: Combination with other FL methods in heterogeneous distribution settings.

We verify the applicability of FedFit to other FL methods. The experimental results are shown in Figure 4, where FedProx adds proxy term loss during local training, and FedAdam conducts adaptive weight updates after server aggregation. It is known that FedProx and FedAdam are more efficient methods than FedAvg and improve model performance. FedFit is an aggregation method to replace averaging aggregation with linear regression (along with model compression). It does not matter how local training is conducted or what additional computation is run after model aggregation in the server. Therefore, as the figure shows, FedFit is compatible with other recently proposed FL methods, and it is possible to use them in combination to improve model performance.

7 Conclusion

In this paper, we proposed FedFit, which is the reformulation for server aggregation of Federated Learning (FL). We considered the connection between linear regression and FL, and FedFit enables us to utilize the established linear regression techniques in FL. As an example of the applications of linear regression techniques for FL, we demonstrate the experiments that FedFit with robust regression can protect the global model against attacks from collapsed clients and FedFit with sparse regression can reduce the global model size for a reduction of communication cost. In addition, the reformulation of FedFit naturally reduces the amount of data traffic to upload a model because clients upload the compressed model parameters to the server. We believe FedFit has much potential for further research. Although we demonstrated the utilization of the two techniques in the experiments, other advanced techniques presented in the linear regression are also applicable to the reformulation of FedFit. In future work, we plan to utilize the benefits of advanced regression methods in other FL scenarios.

REFERENCES

Eugene Bagdasaryan, Andreas Veit, Yiqing Hua, Deborah Estrin, and Vitaly Shmatikov. How to backdoor federated learning. PMLR, 2020.

Arjun Nitin Bhagoji, Supriyo Chakraborty, Prateek Mittal, and Seraphin Calo. Analyzing federated learning through an adversarial lens. ICML, 2019.

Ella Bingham and Heikki Mannila. Random projection in dimensionality reduction: applications to image and text data. ACM SIGKDD, 2001.

K. A. Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H. Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for federated learning on user-held data. In *NIPS Workshop on Private Multi-Party Machine Learning*, 2016.

Xiaoyu Cao, Minghong Fang, Jia Liu, and Neil Zhenqiang Gong. Fltrust: Byzantine-robust federated learning via trust bootstrapping. NDSS, 2021.

Sanjoy Dasgupta and Anupam Gupta. An elementary proof of a theorem of johnson and lindenstrauss. *Random Structures & Algorithms*, 2003.

Enmao Diao, Jie Ding, and Vahid Tarokh. Heterofl: Computation and communication efficient federated learning for heterogeneous clients. ICLR, 2021.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. ICLR, 2020.

Sannara Ek, François Portet, Philippe Lalanda, and Germán Vega. A federated learning aggregation algorithm for pervasive computing: Evaluation and comparison. In *19th IEEE International Conference on Pervasive Computing and Communications*. IEEE, 2021.

Alireza Fallah, Aryan Mokhtari, and Asuman Ozdaglar. Personalized federated learning: A meta-learning approach. Arxiv, 2020.

Clement Fung, Chris J.M. Yoon, and Ivan Beschastnikh. Mitigating sybils in federated learning poisoning. Arxiv, 2018.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. ICLR, 2015.

Jakub Konečný, H. Brendan McMahan andFelix X. Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. Federated learning: Strategies for improving communication efficiency. NIPS Workshop on Private Multi-Party Machine Learning, 2016.

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

David Leroy, Alice Coucke, Thibaut Lavril, Thibault Gisselbrecht, and Joseph Dureau. Federated learning for keyword spotting. ICASSP, 2019.

Ang Li, Jingwei Sun, Binghui Wang, Lin Duan, Sicheng Li, Yiran Chen, and Hai Li. Lotteryfl: Personalized and communication-efficient federated learning with lottery ticket hypothesis on non-iid datasets. Arxiv, 2020a.

Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. Federated learning: Challenges, methods, and future directions. IEEE Signal Processing Magazine, 2020b.

Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. ICLR, 2014.

Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, 2011.

Odalric-Ambrym Maillard and Rémi Munos. Linear Regression with Random Projections. *Journal of Machine Learning Research*, 13:2735–2772, 2012.

H. Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. AISTATS, 2017.

Mehryar Mohri, Gary Sivek, and Ananda Theertha Suresh. Agnostic federated learning. ICML, 2019.

Titouan Parcollet, Javier Fernandez-Marques, Pedro Pb Gusmao, Yan Gao, and Nicholas D. Lane. Zerofl: Efficient on-device training for federated learning with local sparsity. ICLR, 2022.

Jeffrey Pennington, Richard Socher, and Christopher Manning. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Doha, Qatar, October 2014. Association for Computational Linguistics.

Sashank J. Reddi, Zachary Charles, Manzil Zaheer, Zachary Garrett, Keith Rush, Jakub Konečný, Sanjiv Kumar, and Hugh Brendan McMahan. Adaptive federated optimization. ICLR, 2021.

Amirhossein Reisizadeh, Aryan Mokhtari, Hamed Hassani, and Ramtin Pedarsani Ali Jadbabaie. Fedpaq: A communication-efficient federated learning method with periodic averaging and quantization. AISTATS, 2020.

Karan Singhal, Hakim Sidahmed, Zachary Garrett, Shanshan Wu, Keith Rush, and Sushant Prakash. Federated reconstruction: Partially local federated learning. NeurIPS, 2021.

Ziteng Sun, Peter Kairouz, Ananda Theertha Suresh, and H. B. McMahan. Can you really backdoor federated learning? the 2nd International Workshop on Federated Learning for Data Privacy and Confidentiality at NeurIPS, 2019.

Ilya Tolstikhin, Neil Houlsby, Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Andreas Steiner, Daniel Martin Keysers, Jakob Uszkoreit, Mario Lučić, and Alexey Dosovitskiy. Mlp-mixer: An all-mlp architecture for vision. In *NeurIPS*, 2021.

Tiffany Tuor, Shiqiang Wang, Bong Jun Ko, Changchang Liu, and Kin K. Leung. Overcoming noisy and irrelevant data in federated learning. ICPR, 2020.

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. arXiv, 2017.

Howard H. Yang, Zuozhu Liu, Tony Q. S. Quek, and H. Vincent Poor. Scheduling policies for federated learning in wireless networks. IEEE Transactions on Communications, 2019.

Jie Zhang, Song Guo, Xiaosong Ma, Haozhao Wang, Wencao Xu, and Feijie Wu. Parameterized knowledge transfer for personalized federated learning. NeurIPS, 2021.

Xiang Zhang, Junbo Jake Zhao, and Yann LeCun. Character-level convolutional networks for text classification. In *NIPS*, 2015.

A Appendix

A.1 Dataset

In the experiments and analysis, we used the following dataset.

CIFAR-10/CIFAR-100 have 50,000 training data and 10,000 test data, with 10/100 classes, respectively (Krizhevsky, 2009).

Fashion-MNIST has 60,000 training data and 10,000 test data with 10 classes (Xiao et al., 2017).

IMDB has 25,000 training data and 25,000 test data with 2 classes (Maas et al., 2011).

AGNews has 120,000 training data and 7,600 test data with 4 classes (Zhang et al., 2015).

Yahoo Answers has 1,400,000 training data and 60,000 test data with 10 classes (Zhang et al., 2015).

A.2 Architecture

We used the following architectures for CNN and LSTM in this paper.

CNN. A CNN comprises six convolution layers with a 3x3 kernel (the output channels are 64, 128, 256, 256, 512, and 512, respectively). The max-pooling with a 2x2 kernel is followed after the first, second, and fourth convolution layers, and one fully-connected layer with the global average pooling (Lin et al., 2014) is applied after the final output of the convolution layer. ReLU activation function is used after the operation of each convolution.

LSTM. An LSTM comprises one layer, where the hidden state has 256 dimensions. For the input to the LSTM layer, we use a pretrained word embedding model, GloVe (Pennington et al., 2014), with 300 output dimensions in this paper. For the output layer, we use one fully-connected layer.

A.3 Hyperparameters

A.3.1 In experiments

In the experiments (Sec. 5), we used the following hyperparameters. We set a learning rate (SGD optimizer) as 0.01 with a momentum of 0.9 during local training. During server aggregation (Adam optimizer), we set a learning rate as 1×10^{-4}, β_1 and β_2 are set as 0.9 and 0.999. We set local epochs to 5. These parameters are used in both CNN and LSTM experiments except for collaboratively attacking scenarios in robust experiments, where collapsed clients set a learning rate as 0.1. For norm bound B used in the clip norm defense method (Sun et al., 2019) in attacking experiments, we set B as 10. As compression during model uploading from a client, 90% of dimensions are compressed by the matrix product in each layer.

A.3.2 In analysis

Analysis in the tradeoff setup. In the tradeoff analysis (Sec. 6.1), we set the same hyperparameters used in the experiments.

Analysis in the heterogeneous setup. The same hyperparameters are used in server aggregation in the heterogeneous setup (Sec. 6.2). For local training and other parameters used in FedProx and FedAdam, we set different parameters based on the task. We summarize hyperparameters in Table 3.

Table 3: Summary of hyperparameters in the analysis of heterogeneous settings.

	CNN	LSTM
local epoch	5	1
local learning rate	0.01	0.1
FedProx proxy term	0.1	0.01
FedAdam τ	1×10^{-3}	1×10^{-3}
FedAdam learning rate	0.01	0.005
FedAdam betas	(0.9, 0.99)	(0.9, 0.99)

A.4 Setup in analysis

A.4.1 tradeoff setting

In the tradeoff analysis (Sec. 6.1), output channels are compressed in each layer regardless of the architecture. For the image classification task in CNN, we use CIFAR-100 dataset. We split the training dataset into 50 subsets and assigned one of them to each client. For the text classification task in LSTM, we use IMDB. We split the training dataset into 100 subsets and assigned one of them to each client.

A.4.2 heterogeneous distribution setting

To simulate a heterogeneous distribution setup (Sec. 6.2), we manually create a heterogeneous setup by splitting the training dataset of CIFAR-10 and Yahoo Answers.

For CIFAR-10, we split the training dataset into 100 subsets, each containing only two out of ten classes, and assigned one to each client. The total communication rounds are 500, and the server randomly chooses 20 clients in one communication round. During model uploading from a client, clients compressed the kernel weights and bias in the convolutional layer and output channels in the fully-connected layer. As compression, 50% of dimensions are compressed after the matrix product in each layer.

For Yahoo Answers, we split the training dataset into 500 subsets, each subset including only two out of ten classes, and assigned one to each client. We set 300 communication rounds, and the server randomly chooses 10 out of clients in one communication round. The architecture used in this analysis is LSTM, so output channels are compressed during model uploading from a client. As compression, 90% of dimensions are compressed by the matrix product in each layer.