Information Sciences 560 (2021) 493–503

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

MHAT: An efficient model-heterogenous aggregation training scheme for federated learning

Li Hu [a,b,c], Hongyang Yan [b,*], Lang Li [b,*], Zijie Pan [b], Xiaozhang Liu [a], Zulong Zhang [b]

[a] *School of Computer Science and Cyberspace Security, Hainan University, Hannan, PR China*
[b] *School of Artificial Intelligence and Blockchain, Guangzhou University, Guangzhou, PR China*
[c] *Peng Cheng Laboratory, Shenzhen, PR China*

ARTICLE INFO

Article history:
Received 3 September 2020
Received in revised form 12 January 2021
Accepted 16 January 2021
Available online 2 February 2021

Keywords:
Collaborative learning
Knowledge distillation
Heterogeneous
Communication

ABSTRACT

Federated Learning allows multiple participants to jointly train a global model while guaranteeing the confidentiality and integrity of private datasets. However, current server aggregation algorithms for federated learning only focus on model parameters, resulting in heavy communication costs and low convergence speed. Most importantly, they are unable to handle the scenario wherein different clients hold different local models with various network architectures. In this paper, we view these challenges from an alternative perspective: we draw attention to what should be aggregated and how to improve convergence efficiency. Specifically, we propose MHAT, a novel model-heterogenous aggregation training federated learning scheme which exploits a technique of Knowledge Distillation (KD) to extract the update information of the heterogenous model of all clients and trains an auxiliary model on the server to realize information aggregation. MHAT relaxes clients from fixing on an unified model architecture and significantly reduces the required computing resources while maintaining acceptable model convergence accuracy. Various experiments verify the effectiveness and applicability of our proposed scheme.

1. Introduction

Federated learning [1–3] is an emerging machine learning paradigm for decentralized data [4,5], which enables multiple parties to collaboratively train a global model without sharing their private data. In the canonical federated learning protocol [6], model parameter is the interactive information between clients and the server. At the beginning of each training round, the central server distributes current global model to all participants, and all participants update the local model using their private data. Then the server will collect updated model information from all parties and obtain the central global model via a weighted average aggregation algorithm.

However, in a practical federated learning, several major challenges still remain unsolved when using model parameters as interactive information: First, current aggregation methods require a uniform architecture across all local models. Actually, in real scenarios, clients in federated learning may choose different model architectures due to different training objectives [7,8]. Second, the increasing scale of both clients and model size have resulted in the huge expansion of communication overhead, which significantly decreases the efficiency of federated aggregation. Third, a participant whose data is insufficient for a well performed local model can obtain a better one through federated learning with the model parameters of other

* Corresponding authors at: School of Artificial Intelligence and Blockchain, Guangzhou University, Guangzhou, PR China.
E-mail address: hyang.yan@foxmail.com (H. Yan).

https://doi.org/10.1016/j.ins.2021.01.046
0020-0255/© 2021 Elsevier Inc. All rights reserved.

participants. But for those participants who have sufficient data, their benefits of participating federated learning are still an open problem.

To address above challenges, existing works use the output of the model as interactive information. Li et al. [9] use a more transparent framework based on knowledge distillation to make communications between clients, which allows clients design their model architectures uniquely. In [10], Jeong et al. propose a distributed model training algorithm called Federated Distillation (FD) in order to reduce the communication overhead between devices. Its communication payload is much smaller than the scheme Federated Learning (FL) [1], especially when the model size is relatively large. Chang et al. [11] prove that the usage of high-dimensional model parameters as interactive information is limited to the models with homogeneous architectures, and this interactive method is particularly vulnerable to the privacy and adversarial attacks. In this regard, they propose a robust collaborative machine learning framework called Cronus through using robust knowledge transfer between the clients' black box local models, which can be used to control, unify and significantly reduce the dimensionality of information exchanged. However, none of these methods present effective analysis to guarantee the aggregation result is the optimal one. Because works [9,10] use the method of averaging the logit distribution, and work [11] designs an robust mean estimation algorithm for aggregation. These trivial aggregation methods will influence model convergence.

Inspired by these works, in this paper, we propose a novel scheme, which lets the logit distribution of each client be aggregated by itself. More details, a model will be trained on the server to aggregate the updated information from each client. Experiments have proved that our aggregation method significantly reduces communication consumption in the federated learning while maintaining high model accuracy. Futhermore, the proposed scheme also improves the speed of model convergence.

The main contributions of this work are summarized as follows:

- We propose a novel federated learning scheme for heterogeneous model architectures called MHAT. We focus on model output instead of model parameters and leverage techniques from Knowledge Distillation, which could tackle the challenge of model heterogeneous, and reduce the client's resource consumption.
- We train an auxiliary model on the server to realize information aggregation, which significantly improves the speed of model convergence while maintaining acceptable model convergence accuracy. To the best of our knowledge, MHAT is the first scheme aiming at obtaining optimal aggregation results.
- We conduct various experiments to verify the effectiveness of MHAT, including investigating its performance on different numbers of clients, different combinations of hyperparameters and different datasets. We also compare MHAT with other state-of-the-art methods and present sufficient analysis for our experiments.

1.1. Organization

The remainder of this paper is organized as follows. Section 2 gives the preliminary knowledge of federated learning, knowledge distillation, and federated distillation. In Section 3, we present our detailed designs of mutual federated learning framework. The system evaluation and experimental results are presented in Section 4. Section 5 discusses the related works and Section 6 concludes this paper.

2. Preliminaries

2.1. Federated learning

Federated learning enables multiple data holders to jointly train a global model by only sharing their training gradients or parameters. In this way, each client ultimately obtains a model that performs much better than the one trained on its own local data. The Federated Averaging (FedAvg) [1] algorithm provides a general description of the original federated learning process.

In a federated learning scenario, n clients negotiate and cooperate to train a model, then send the model structure to the central server. The server initializes the model parameters θ and distributes them to each user. Client $k(k = 1, 2, \dots, n)$ processes a dataset D_k containing N_k samples $X_k = \{x_{ki}\}_{i=1}^{N_k}$, of the M class, with the corresponding label set of $Y_k = \{y_{ki}\}_{i=1}^{N_k}, y_{ki} \in \{1, 2, \dots, M\}$. After obtaining the global model, each user uses local data and gradient descent optimization algorithm to update it, then obtains new model parameters θ_k, $\theta_k = \theta - \bigtriangledown L(D_k; \theta)$, where $L(D_k; \theta)$ is a loss function on the dataset D_k with respect to θ, and α is the learning rate. Then these locally updated parameters are uploaded to the server for aggeration using the formula as follows:

$$\theta^{global} = \sum_{k=1}^{n} \frac{N_k}{\sum_{k=1}^{n} N_k} \theta_k. \tag{1}$$

Until the global model converges, the server broadcasts the final updated model parameter θ^{global} to each client.

2.2. Knowledge distillation

Knowledge distillation was first proposed in [12]. Its purpose is to transfer the knowledge learned from a complex model or multiple models (as teacher model) to another lightweight model (as student model), while making the model as light as possible without loss of performance. Here, knowledge should be understood broadly and abstractly. Model parameters, network output from both hidden layers and output layer (features extracted by the network), model perdition results, etc. can all be considered as knowledge.

According to the type of knowledge being transferred, knowledge distillation can mainly be divided into three categories: Output Transfer [12–14] – using the output of the network as knowledge, Feature Transfer [15,16] – using the characteristics of the network learning (learning the output of the hidden layer – Feature map) as knowledge, and Relation Transfer [17] – taking the relationship of the network or sample as knowledge. Taking the output of the network as knowledge for an example, in the process of knowledge distillation, the teacher model trains the model with all data and uses a small subset of the data when training the student model. A well trained student model learns not only the label of the data (called hard label), but also the predictive distribution of this part of the data output by the teacher model (called soft labels), which is generated using the formula as follows:

$$p_i = \frac{exp(z_i/T)}{\sum_{j=1}^{M} exp(z_j/T)} (i = 1, 2, \ldots, M), \tag{2}$$

where z_i represents the value corresponding to the i-th category output by the last layer of the model, and T is called temperature, which smoothes the probability distribution of the output. The appearance of soft labels greatly improves the accuracy of student model training, which means that soft labels convey the knowledge learned during the training of the teacher model.

Existing related works introduce the training method of knowledge distillation to improve the original federated learning, and change the model parameters as the interactive information between clients and the server to the result of the client's predicted output. Related papers show that federated learning in this way can also achieve acceptable performance, but only by performing a weighted average of the output results uploaded by each client. Therefore, we try to improve the aggregation process of the probability distribution of each client's output.

2.3. Federated distillation

Federated learning is a system that allows participants to build virtual common models of data and then benefit from it, while preserving data locally and not compromising privacy or regulations. In the original federated learning framework, a global model with one specific architecture is updated through interaction of the model parameters. However, authors in FD [10] points out that when the model size is large, there will be huge communication overhead if the model parameters are used as interactive information. The article FedMD [9] points out that the use of model parameters as interactive information will limit the application of model architecture. In order to solve these problems, they perform federated distillation learning, citing the idea of knowledge distillation to use the probability distribution output by the model as the interaction information between the user and the server. Fig. 1 gives a high-level overview of KD. We briefly describe it below.

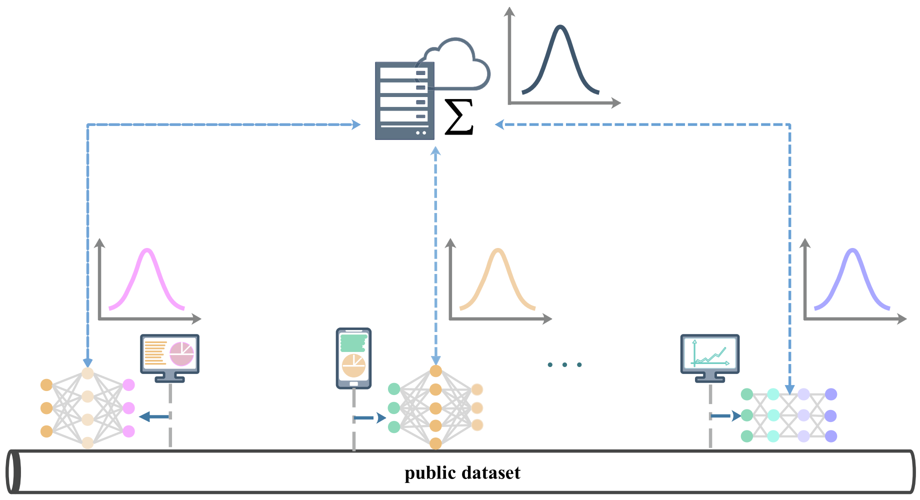

Fig. 1. The illustration of FD framework.

In a federated distillation framework with n-parties, each client initializes its own model parameter $\theta_k (k \in \{1, 2, \ldots n\})$, and the models are heterogeneous between clients, that is, they have different model structures and sizes. User k has a dataset D_k containing N_k samples $X_k = \{x_{ki}\}_{i=1}^{N_k}$, of the M class, with the corresponding label set of $Y_k = \{y_{ki}\}_{i=1}^{N_k}, y_{ki} \in \{1, 2, \ldots, M\}$, at the same time, each client has a dataset D containing N samples $X = \{x_i\}_{i=1}^{N}$, of the M class, with the corresponding label set of $Y = \{y_i\}_{i=1}^{N}, y_i \in \{1, 2, \ldots, M\}$. Each client updates the model parameters based on their local datasets D_k and D, and obtains $\theta_k = \theta_k - \bigtriangledown L(D_k \bigcup D; \theta_k)$, where $L(D_k \bigcup D; \theta_k)$ is about θ_k loss function on the dataset $D_k \bigcup D$, and α is the learning rate. Each party then performs prediction on dataset D based on locally updated model and uploads the result $\overline{Y}_k = f(\theta_k, X)$ to the server, $f(\theta_k, X)$ represents the probability distribution of input X passing through the model with parameters θ_k. Then, the server uses the formula as follows to aggregate these probability distributions to obtain a new predictive distribution of public dataset D:

$$\widetilde{Y} = \frac{1}{n} \sum_{k=1}^{n} \overline{Y}_k, \tag{3}$$

Each client obtains $\widetilde{Y}$ through the server broadcast, and updates their local model through the dataset $D_k \bigcup D \bigcup \left(X, \widetilde{Y}\right)$.

3. The detail for federated learning model-heterogenous aggregation training

The framework of MHAT is presented in this section. We first give the overview of MHAT. Then we discuss the designed algorithm in two stages: (1) client execution training and prediction, (2) server execution training and prediction.

3.1. Overview

The overview of MHAT is given in Fig. 2. Previous aggregation algorithms mainly relied on weighted average of the information uploaded by the client to obtain global information. We argue that this choice is not optimal, especially for using probability distribution as interactive information between the client and the server. Instead, we propose to train an auxiliary model on the server to aggregate information uploaded by the client. Our scheme aims to make the aggregation more sufficient while allowing participants to independently design their model architectures, which makes the aggregation result better, improves the model convergence speed, and reduces the interaction between clients and the server. At the same time, it is guaranteed that the performance of the model obtained by the each participant is far better than the model trained only on its own data.

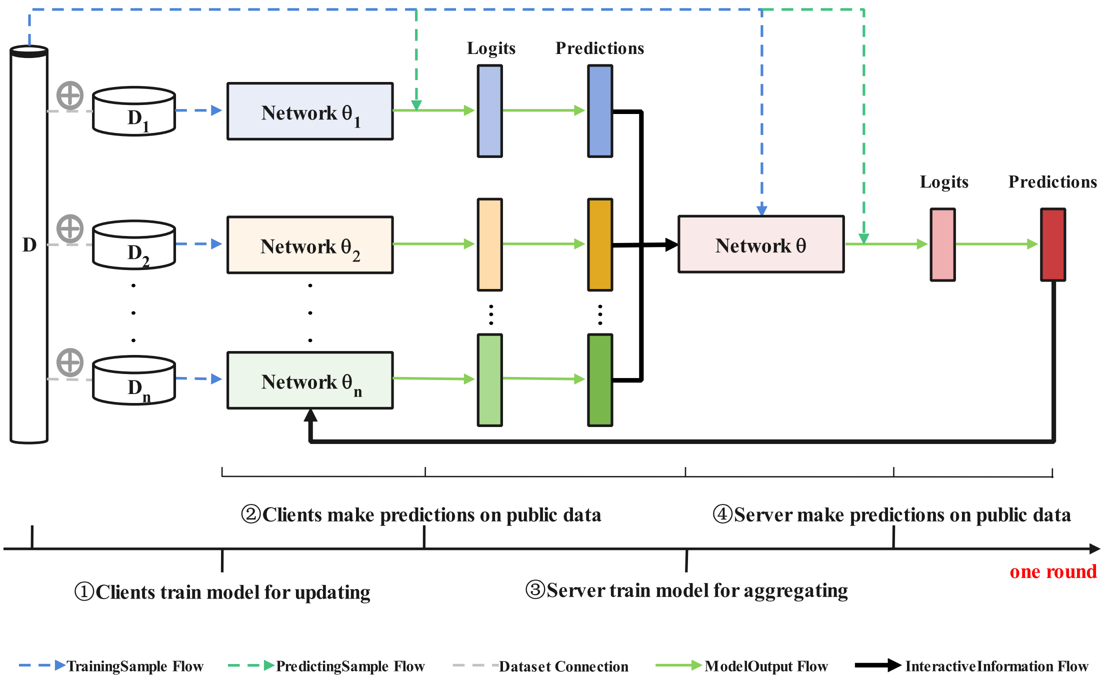
Logits
Predictions
D
D1
D2
Dn
Network θ1
Network θ2
Network θn
Logits
Predictions
Network θ
②Clients make predictions on public data
④Server make predictions on public data
①Clients train model for updating
③Server train model for aggregating
one round
TrainingSample Flow
PredictingSample Flow
Dataset Connection
ModelOutput Flow
InteractiveInformation Flow

Fig. 2. The illustration of MHAT framework.

In this scheme, we design two parts of dataset: private dataset and public dataset. Each client has its own private dataset and public dataset as training samples, and server just possesses public dataset. The entire interaction process includes four steps. First, each client independently selects its own network architecture, and uses training samples for model training. Second, each client uploads the predictive distribution of their update model on the public data set to the server. Third, an auxiliary model is trained on the server to aggregate the probability distribution uploaded by each client. Finally, a prediction is performed on the public data using the auxiliary model, whose results are then broadcasted to each client. After that, each client updates its training samples by adding the probability distribution corresponding to the public data sent by the server, and then returns to the first step to repeat the whole process until the average accuracy of each local model converges.

Algorithm 1. Model-Heterogenous Aggregation Training for Federated Learning(MHAT)

```
Input: $D_k$ : private dataset for each client k, $(k \in \{1, 2, \ldots, n\})$
$D$: public dataset for all clients and server
Output: $\theta_k^t$: model parameters for each client k
Initialize:
each client initialize $\theta_k^0$ by themself
server initialize $\theta^0$ by all clients
for each round $t = 0, 1, 2, \ldots$ do
if $t > 0$ then
$\theta^t, \widetilde{Y}^t = ServerUpdate\left(\theta^{t-1}, \overline{D}, D\right)$    // $\widetilde{Y}^t$ will be broadcast to each client
end if
$\overline{D} = D$
for each client $k = 1, 2, \ldots, n$ do
if $t == 0$ then
$\widetilde{D}_k = D_k \bigcup D$
else
$\widetilde{D}_k = D_k \bigcup D \bigcup \left(X, \widetilde{Y}^t\right)$
end if
$\theta_k^{t+1}, \overline{Y}_k^t = ClientUpdate\left(\theta_k^t, \widetilde{D}_k, D\right)$    // $\overline{Y}_k^t$ will be uploded to server
$\overline{D} = \overline{D} \bigcup (X, \overline{Y}_k^t)$
end for
end for
```

3.2. Detailed operations

Algorithm 1 describes our proposed scheme(called MHAT) in detail. Here, we use D to represent the public dataset, and D_k to represent the private dataset of user k $(k \in \{1, 2, \ldots, n\})$, which is consistent with the definition of Federated Distillation in Section 2.3. Client k initializes its model parameter θ_k^0 according to their own requirements, and with all client agreements, server chooses a auxiliary model and randomly initializes the auxiliary model parameter θ^0. It will be our follow-up research work on how to select the model architecture of the server to make the aggregation effect optimal.

Algorithm 2. Client Update

```
Input:
$\theta$: model parameters before update
$\widetilde{D}$: training dataset
$D$: public dataset
Output:
$\theta$: model parameters after update
$\overline{Y}$: predictive distribution of model output on public dataset
Initialize:
$\alpha_1$: the learning rate of client traning
```

(continued on next page)

```
B_1: the training minibatch size
E_1: the number of training epochs
function ClientUpdate($\theta, \tilde{D}, D$):
    $B \leftarrow$ (split $\tilde{D}$ into batches of size $B_1$)
    for each epoch i from 1 to $E_1$ do
        for batch b $\in B$ do
            $\theta = \theta - \alpha_1 \bigtriangledown L\left(\tilde{D}; \theta\right)$
        end for
    end for
    $\overline{Y}$= f($\theta$, X)
    return $\theta, \overline{Y}$
end function
```

3.2.1. Client update

Algorithm 2 describes the update process of client side. Client k updates model parameters with dataset $\tilde{D}_k = D_k \bigcup D \bigcup \left(X, \tilde{Y}^t\right)$, and performs a prediction on dataset D, then sends the prediction result $\overline{Y}_k^t$ to the server. In *t-th* round, the loss function of client k is as follows:

$$L\left(\tilde{D}_k; \theta_k^t\right) = H\left(Y_k, P^1\right) + H\left(Y, P^2\right) + H\left(\tilde{Y}^t, P^2\right) = -\left(\sum_{i=1}^{N_k} y_{ki} logp_{ki}^1 + \sum_{i=1}^{N} y_i logp_{ki}^2 + \sum_{i=1}^{N} y_i^t logp_{ki}^2\right) \quad (4)$$

where $P^1 = f(\theta_k^t, X_k), P^2 = f(\theta_k^t, X), f(\theta_k^t, \cdot)$ represents the probability distribution of the input passing through the model with parameters θ_k^t. Then the model parameter updated by client k is

$$\theta_k^{t+1} = \arg\min_{\theta_k^t} L\left(\tilde{D}_k; \theta_k^t\right) \quad (5)$$

3.2.2. Server update

Algorithm 3 describes the update process of server side. After collecting the interactive information uploaded by each client, the server updates its model based on the interactive information, so as to aggregate the interactive information, then predicts the public dataset, and returns the prediction result to the client for updating. The server updates model parameters with dataset $\overline{D} = D \bigcup \left(\bigcup_{k=1}^{n} (X, \overline{Y}_k^t)\right)$, and performs a prediction on dataset D, then broadcasts the prediction results $\tilde{Y}^{t+1}$ to each client. In $(t+1)$-*th* round, the loss function of server is as follows:

$$L(\overline{D}; \theta^t) = H(Y, P) + \sum_{k=1}^{n} H(\overline{Y}_k^t, P) = -\left(\sum_{i=1}^{N} y_i logp_i + \sum_{k=1}^{n}\sum_{i=1}^{N} y_{ki}^t logp_{ki}\right) \quad (6)$$

where $P = f(\theta^t, X), f(\theta^t, \cdot)$ represents the probability distribution of input X passing through the model with parameters θ_k^t, then the model parameter updated by server is

$$\theta^{t+1} = \arg\min_{\theta^t} L(\overline{D}; \theta^t) \quad (7)$$

Algorithm 3. Server Update

```
Input:
$\theta$: model parameters before update
$\overline{D}$: training dataset
$D$: public dataset
Output:
$\theta$: model parameters after update
$\tilde{Y}$: predictive distribution of model output on public dataset
```

```
Initialize:
$\alpha_2$: the learning rate of server traning
$B_2$: the training minibatch size
$E_2$: the number of training epochs
function ServerUpdate($\theta, \overline{D}, D$):
    $B \leftarrow$ (split $\overline{D}$ into batches of size $B_2$)
    for each epoch i from 1 to $E_2$ do
        for batch b $\in B$ do
            $\theta = \theta - \alpha_2 \bigtriangledown L(\overline{D};\theta)$
        end for
    end for
    $\widetilde{Y}$= f($\theta$, X)
    return $\theta, \widetilde{Y}$
end function
```

4. Performance evaluation

In this section, we evaluate the performance of our scheme in image classification tasks with common ML models over MNIST datasets [18]. We demonstrate the effectiveness of our approach by comparing it with the ideal baseline mechanisms FD. Our experiments are implemented with Tensorflow [19].

In our experiments, we consider a typical FL scenario in which a server coordinates multiple clients. In order to verify the effectiveness of our scheme, we first select a set of suitable hyperparameters on the experiments of 3 clients as the basis for subsequent experiments, and verify the influence of the number of clients on the experiment. Then we compare it with the baseline experiment. Here, we compare the effects of different numbers of clients and whether there are tags in the public dataset on the experiment. Finally, we test the performance with heterogeneous model.

4.1. Datasets and ML models

For the image classification tasks, we use MNIST dataset that is a widely used benchmark in the FL literature [20]. Then we choose simple fully connected neural network to verify MHAT.

The MNIST dataset includes 60,000 training samples and 10,000 testing samples. Following [1], we sort data samples based on shuffling the training dataset. We first divide the training data set into a private dataset and a public dataset. Considering whether there are labels in the public data set, we have divided into two proportions. When the public dataset has labels, we divide 2000 training data into public dataset, and each client has 2900 training data as private dataset. When the public dataset is unlabeled, we divide 10,000 training data into the public dataset, and each client has 2000 training data as a private dataset. We will analyze why the dataset is divided in this way in the results section.

The model architecture is taken from the tensorflow tutorial [19], in order to reflect the heterogeneity of the model, in this paper we set up neural networks with different layers and different numbers of neurons. To verify the effectiveness of our schemes, we first make a comparison under the setting of model homogeneity, and then further expand to compare the scheme with heterogeneous models. In the setting with homogeneous model, we choose a two-layer fully connected network (the first with 256 units, the second with 64 filters, each followed with tanh activation function) with a softmax output layer for analysis, This probably has $2 * 10^5$ parameters in total. At this time, the model architectures of the client and the server are the same. In the setting with heterogeneous model, the number of layers of each client and server model and the number of neurons in each layer are different. In this paper, we set two choices for the number of layers of the model: 2 layers or 3 layers. For the number of neurons in each layer, we choose it from the set 32, 64, 128, 256, 512, 1024.

4.2. Result analysis

4.2.1. Basic experiment

4.2.1.1. Select hyperparameters. In this part, we test the effects of initialization training epoch E1, server aggregation training epoch E2, client local updating epoch E3, temperature T and learning rate β of distillation process. We train 3 local models and one server model with the same neural network and labeled public dataset in the IID setting. Based on the selection of training hyperparameters of original federated learning, we set the experimental hyperparameters to the following values: E1 = 5, E2 = 1, E3 = 1, T = 1, β = 0.5. The influence of different hyperparameters is tested through fixing other variables. Fig. 3 describes the impact of different hyperparameter on the test accuracy. According to the experimental results, we choose E1 = 5, E2 = 5, E3 = 1, T = 5, β = 0.5 as optimal setting values, as it enables the clients to train locally with fewer times and

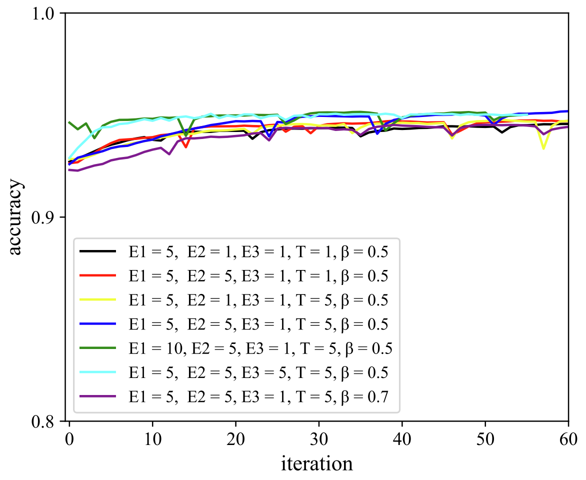

Fig. 3. Top-1 accuracy corresponding to client model under different hyperparameters.

achieve the highest test accuracy. In the following, we will further validate our proposed scheme with this set of hyperparameters.

4.2.1.2. The impact of the number of clients. Under the selected hyperparameters, we report the test accuracy of 3 clients and 20 clients with labeled public data and same model architecture. As shown in Fig. 4, the trend of results is consistent with the original federated learning, and the increase of the number of clients would improve the performance of federated learning.

The above experiments show that it is feasible to train a model on the server side to realize the aggregation of the information uploaded by the clients. It can meet the convergence trend of federated learning and meet the influence of the number of clients on the convergence accuracy of federated learning.

4.2.2. Comparison experiment

In this part, in order to further verify the effectiveness of our scheme, we compare the results of federated distillation(FD) with different settings. In the training process of federated distillation, the predictive distribution uploaded by each client will be weighted averaged. First, we train 20 clients with the same neural network and labeled public data set, then record the test accuracy of the MHAT and FD schemes respectively, as shown in Fig. 5, and we find that MHAT could also achieve the same optimal accuracy.

Considering that when the public dataset has their labels, the client and the server will also learn about the labels of the public data when updating the model, which will have a certain impact on the learning of interactive information and will affect the comparison of MHAT and FD experiments. So we choose to perform a comparative test on an unlabeled public dataset. We report the test accuracy of 3 clients and 20 clients with unlabeled public data and the same neural network.

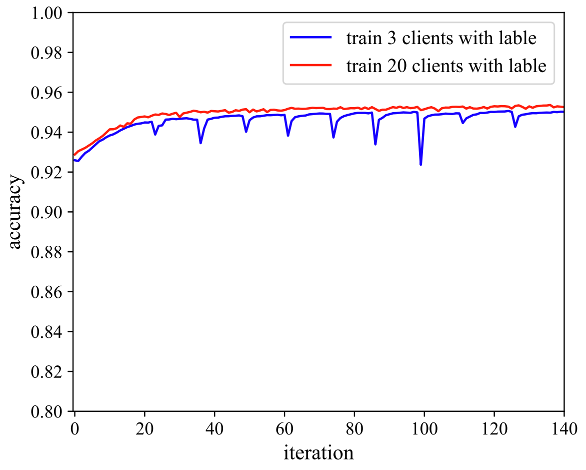

Fig. 4. Comparing the MHAT scheme results between 3 clients and 20 clients using tagged public data.

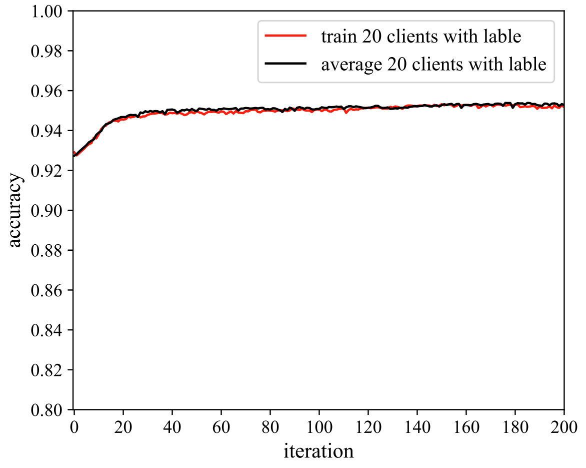

Fig. 5. The results of MHAT scheme and FD scheme with 20 clients using labeled public data.

We can see in Fig. 6, MHAT and FD schemes still have similar convergence effects. At the same time, we found that when our scheme achieves the optimal accuracy, while the number of rounds required for interaction is much less than that of the federated distillation solution. Table 1 lists the specific differences. It can be seen that when the number of clients is 3 and the public data is labeled, our scheme only requires 62 rounds of interaction to achieve the best test accuracy of 95.19%, while the federated distillation scheme requires 188 rounds of interaction to achieve the best test accuracy of 95.29%.

This verifies that with less accuracy loss, the number of interactions required by our solution is 1/3 of the FD solution, which can greatly reduce the resource consumption of clients. We also compare the unlabeled case of the public datasets. The experimental results show that the number of interactions required by our solution is 1/2 of the FD solution when the public datasets is not labeled with acceptable accuracy loss.

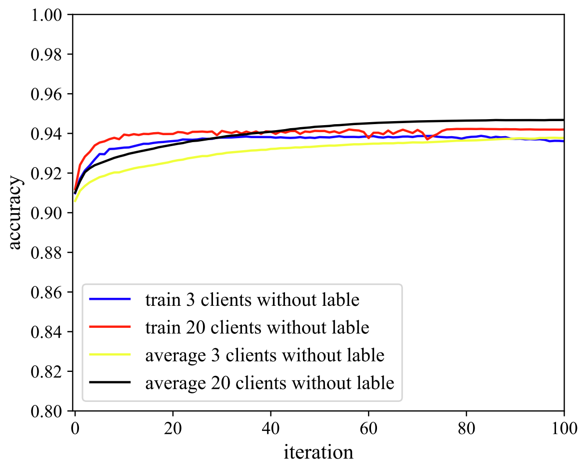

Fig. 6. Convergence behavior about the MHAT scheme and the FD scheme using unlabeled public data to train 3 and 20 clients.

Table 1
The number of interactions between clients and server when the model reaches the Top-1 accuracy (%) on the mnist dataset. FD – MHAT measures the difference in the number of interactions and accuracy between the FD and MHAT.

Number of clients	With Lable			Without Lable		
	MHAT	FD	FD - MHAT	MHAT	FD	FD - MHAT
3	62(95.19)	188(95.29)	126(0.01)	83(93.88)	170(93.89)	87(0.01)
20	150(95.32)	176(95.38)	26(0.06)	83(94.23)	169(94.72)	86(0.49)

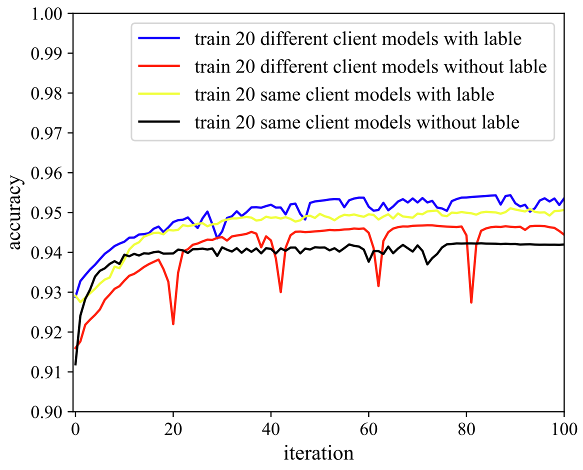

Fig. 7. Use different models to train 3 and 20 clients to compare the convergence behavior of the MHAT scheme with labeled public data and unlabeled public data.

4.2.3. Performance evaluation of heterogeneous models

By comparing with the convergence effect of the model when the model is homogeneous, we further verify the feasibility of our scheme when the model is heterogeneous. We use the labeled public data set and the unlabeled public data set to train 20 clients for testing. We find that changing the model structure can still converge steadily. As shown in Fig. 7, no matter whether the public data set is labeled or not, the model heterogeneous training has the same convergence trend as the model homogeneous training.

5. Related work

Federated learning allows clients to collaborate with each other to get a better model. However, due to the different characteristics of different users, federated learning still faces several challenges, such as system heterogeneity, statistical heterogeneity and model heterogeneity. This paper improves the efficiency of existing schemes by tackling the challenge of Model Heterogeneity.

In the federated learning scenario, the aggregation algorithm plays a key role by combining local model updates from all clients participating in the federated learning. The standard aggregation method for federated learning is FedAvg [1], in which the parameters of local model are averaged according to the amount of data, and the weight is proportional to the size of the client data set. However, authors in [21,22] show that for many tasks, because the accuracy of the global shared model is not as good as the local model trained by themselves, some participants may not get any benefits through participation. Therefore, using this method of averaging weights will have a serious impact on performance, such as communication efficiency.

There are many existing improved aggregation algorithms. FedProx [23] adds a near-end term to the user's local loss function, which limits the impact of local models by restricting their updates to the global model. FedCS [24] manages client devices according to client resource conditions, thereby effectively coping with the problem of client selection with resource constraints. It allows the server to aggregate as many client update information as possible, and accelerates the improvement of the performance of the model. FedMeta [25] shares parameterized algorithms instead of model parameters, which greatly reduces the required communication costs and speeds up convergence. In addition, FedMeta retains user privacy because it only shares parameterized algorithms without data. Federated Matching Average (FedMA) [26], which is mainly based on Probabilistic Federated Neural Matching(PFNM) and model size adaptation, proposes a new layered federated learning algorithm for modern CNNs and LSTMs to achieve better performance and communication efficiency. Agnostic Federated Learning (AFL) [27], as another variant of FedAvg, optimizes the centralized distribution that is a mixture of client distributions. In [28], by modifying the aggregation algorithm, all participants are able to maximize the performance of their local model under a certain global model.

However, the works mentioned above couldn't satisfy the need of client who wants to design their own model. Therefore this paper proposes a new aggregation method, which uses model output as interactive information, realizes information aggregation through model training performed on the server side, and can achieve rapid model convergence in a small number of training rounds. The experiment proves that this scheme can promote the efficiency of federated learning.

6. Conclusion

In this paper, we propose a novel federated learning framework, targeting to achieve high model accuracy and low computing cost in the scenario where clients hold heterogeneous neural network models. Specially, we leverage the techniques from knowledge distillation methods through making model output as knowledge and interacting information between clients and server. Various experiments have verified that our method can achieve excellent performance in both convergence speed and resource consumption.

CRediT authorship contribution statement

Li Hu: Writing - original draft, Methodology, Software, Validation. **Hongyang Yan:** Methodology, Validation, Writing - review & editing. **Lang Li:** Validation, Writing - original draft. **Zijie Pan:** Validation, Writing - original draft. **Xiaozhang Liu:** Writing - review & editing, Supervision. **Zulong Zhang:** Software, Validation.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgement

The research leading to these results has received funding from China Postdoctoral Science Foundation (2020M682658).

References

[1] B. McMahan, E. Moore, D. Ramage, S. Hampson, B.A. y Arcas, Communication-efficient learning of deep networks from decentralized data, in, Artif. Intell. Stat. (2017) 1273–1282.
[2] H.B. McMahan, E. Moore, D. Ramage, B.A. y Arcas, Federated learning of deep networks using model averaging..
[3] Q. Yang, Y. Liu, T. Chen, Y. Tong, Federated machine learning: Concept and applications, ACM Trans. Intell. Syst. Technol. 10 (2) (2019) 1–19.
[4] Y. Zhao, M. Li, L. Lai, N. Suda, V. Chandra, Federated learning with non-iid data..
[5] F. Sattler, S. Wiedemann, K.-R. Mller, W. Samek, Robust and communication-efficient federated learning from non-iid data..
[6] H.B. Mcmahan, E. Moore, D. Ramage, B.A. y Arcas, Federated learning of deep networks using model averaging..
[7] Y. Liu, A. Huang, Y. Luo, H. Huang, Y. Liu, Y. Chen, L. Feng, T. Chen, H. Yu, Q. Yang, Fedvision: an online visual object detection platform powered by federated learning., arXiv: Learning..
[8] O. Choudhury, A. Gkoulalasdivanis, T. Salonidis, I. Sylla, Y. Park, G. Hsu, A. Das, Differential privacy-enabled federated learning for sensitive health data, arXiv: Learning..
[9] D. Li, J. Wang, Fedmd: Heterogenous federated learning via model distillation, arXiv preprint arXiv:1910.03581..
[10] E. Jeong, S. Oh, H. Kim, J. Park, M. Bennis, S.-L. Kim, Communication-efficient on-device machine learning: Federated distillation and augmentation under non-iid private data, arXiv preprint arXiv:1811.11479..
[11] H. Chang, V. Shejwalkar, R. Shokri, A. Houmansadr, Cronus: robust and heterogeneous collaborative learning with black-box knowledge transfer, arXiv preprint arXiv:1912.11279..
[12] G. Hinton, O. Vinyals, J. Dean, Distilling the knowledge in a neural network, arXiv preprint arXiv:1503.02531..
[13] Y. Zhang, T. Xiang, T.M. Hospedales, H. Lu, Deep mutual learning, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 4320–4328.
[14] H. Chen, Y. Wang, C. Xu, Z. Yang, C. Liu, B. Shi, C. Xu, C. Xu, Q. Tian, Data-free learning of student networks, in: Proceedings of the IEEE International Conference on Computer Vision, 2019, pp. 3514–3522.
[15] A. Romero, N. Ballas, S.E. Kahou, A. Chassang, C. Gatta, Y. Bengio, Fitnets: hints for thin deep nets, arXiv preprint arXiv:1412.6550..
[16] J. Wang, W. Bao, L. Sun, X. Zhu, B. Cao, S.Y. Philip, Private model compression via knowledge distillation, in: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 33, 2019, pp. 1190–1197..
[17] J. Yim, D. Joo, J. Bae, J. Kim, A gift from knowledge distillation: fast optimization, network minimization and transfer learning, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017, pp. 4133–4141.
[18] Y. Lecun, L. Bottou, Gradient-based learning applied to document recognition, Proc. IEEE 86 (11) (1998) 2278–2324.
[19] T. team, Tensorflow convolutional neural networks tutorial. URL: http://www.tensorflow.org/tutorials/deep_cnn..
[20] S. Caldas, S.M.K. Duddu, P. Wu, T. Li, J. Konen, H.B. Mcmahan, V. Smith, A. Talwalkar, Leaf: a benchmark for federated settings..
[21] T. Yu, E. Bagdasaryan, V. Shmatikov, Salvaging federated learning by local adaptation (2020). arXiv:2002.04758..
[22] F. Hanzely, P. Richtrik, Federated learning of a mixture of global and local models..
[23] T. Li, A.K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, V. Smith, Federated optimization in heterogeneous networks, arXiv preprint arXiv:1812.06127..
[24] T. Nishio, R. Yonetani, Client selection for federated learning with heterogeneous resources in mobile edge..
[25] F. Chen, M. Luo, Z. Dong, Z. Li, X. He, Federated meta-learning with fast convergence and efficient communication..
[26] H. Wang, M. Yurochkin, Y. Sun, D. Papailiopoulos, Y. Khazaeni, Federated learning with matched averaging, arXiv preprint arXiv:2002.06440..
[27] M. Mohri, G. Sivek, A.T. Suresh, Agnostic federated learning, arXiv preprint arXiv:1902.00146..
[28] T. Li, M. Sanjabi, A. Beirami, V. Smith, Fair resource allocation in federated learning, arXiv preprint arXiv:1905.10497..