Decentralized P2P Federated Learning on Ad-hoc Like Networks with Non-IID Dataset

Qingzhe Jin
The University of Tokyo
howllowkim@g.ecc.u-tokyo.ac.jp

Hideya Ochiai
The University of Tokyo
ochiai@elab.ic.i.u-tokyo.ac.jp

***Abstract*—In the last few decades, Federated Learning (FL) is proposed in order to perform Machine Learning (ML) tasks in a distributed manner while protecting users' privacy and data. However, most of the traditional FL methods rely on centralized entities while in many real-life situations, there isn't any central server which can orchestrate the training procedure. Moreover, it is also easy to be ignored by researchers that the network topology is likely to be changing all the time in some scenarios such as Ad-hoc networks. Besides, how to deal with the unbalanced data which are not independently identically distributed (IID) collected by devices is also an important open problem. As a consequence, in this paper, we propose a peer-to-peer federated learning algorithm with ad-hoc network, along with five model aggregation strategies. We tested our algorithm on self-made Non-IID datasets. After 5000 epochs, the average accuracy of devices reaches 69% under the best strategies on unbalanced CIFAR10 dataset, improving 28% from the self-training cases without P2P communication. The results indicate that our strategies can reduce the negative effect caused by Non-IID datasets even with ad-hoc networks.**

***Index Terms*—decentralized federated learning, ad-hoc network, neural network, non-iid data**

I. Introduction

Federated Learning(FL), a decentralized Machinle Learning method, is firstly introduced by Google research team [1] to train machine learning models without needing to transfer user data onto a central server. In a general federated learning system, every worker has their own dataset with which they can train a local model and send the model updates(gradients, parameters, etc.) to a central parameter server periodically. Then the central server will update the global model and send it back to every worker for synchronization.

Although traditional FL algorithms such as FedAvg have been proved efficient in many scenarios, these algorithms still need a central parameter coordinator, and this may lead to some drawbacks. Firstly, now that a central server is used in an FL setup, there's a potential that one-point failure might happen. Secondly, in many real-life scenarios, there may not be any central node which is connected to every worker in the graph, and the topology of the connection of workers might be changing all the time.

There have been researches about decentralized federated learning, but they're mostly based on fully-connected or static network topology. However, in some dynamic networks, such as Ad-hoc wireless network [2], if a mobile device is shut down, all edges which are connected with it will be deleted from this graph, so the topology is changing all the time. In this paper, we propose a method to simulate the decentralized federated learning procedure on Ad-hoc like dynamic network topology. Moreover, most of the federated learning researches assume that the data collected by every device is independently identically distributed (IID). However, in real-life scenarios, the situation is completely opposite that the data can be severely unbalanced, and it's of vital importance to take the negative effect of training results caused by Non-IID dataset into consideration as well. **The main contribution of this paper** can be summarized as follows:

- Different from existed works on decentralized federated learning using fully connected network topology, we perform the training procedure on an ad-hoc like network.
- In order to simulate the unbalanced dataset in real-life scenarios, we generate Non-IID datasets from CIFAR10 and MNIST, which are used for the training procedure of every device instead of conventional IID datasets. Besides, a special aggregation algorithm is tried to reduce the negative effect of the Non-IID dataset.
- We designed and tried the performance several different communication strategies and aggregation functions for various scenarios.

The paper is organized as follows. We introduce some related works in Section II. In Section III, we describe details of the proposed P2P decentralized federated learning simulation system with ad-hoc like networks. The experimental setup is introduced in Section IV, and the experiment results are discussed and analyzed in Section V. In Section VI, we make a conclusion of the results we've got.

II. Related Work

A. Decentralized Federated Learning

Many researchers have found out drawbacks of conventional centralized federated learning, where devices send the updates of local model parameters to a central node which uses a specific aggregation function to update the global model's parameters and send them back to each worker. To cope with the single point failure problem, decentralized

978-1-6654-6975-3/22/$31.00 ©2022 IEEE

federated learning is proposed. A key part of decentralized FL is how to let workers communicate with each other. Most of the existing works on decentralized federated learning such as [3] use gossiping scheme, which is a widely used communication algorithm in peer-to-peer network, for each worker to spread its model information through the whole network. The authors of [4] use a decentralized P2P file system called IPFS (Interplanetary File system) [5] to let workers exchange model parameters in a way of file sharing. But these work are all based on fully connected networks, which means that every worker can always communicate with any other worker in the network. In order to test the performance of decentralized FL with a dynamic network topology, we use Random Way Point (RWP) model [6], a widely used synthetic model for mobility, to generate the movement track of workers and design an algorithm to simulate the communication and training procedure of the system.

B. Federated learning with Non-IID dataset

In the initial works about FL, a key assumption is that the workers' dataset used for experiments are IID, which leads to extremely satisfying results. However, in fact, the Non-IID data are more often to be seen in most of situations and have become a popular and important open problem in FL. There are mainly two types of solution to this Non-IID problem in existing works. The first one is to rebalance the dataset and change the Non-IID dataset into IID. Jeong et al. use a Generative Adversirial Network(GAN) to reproduce the data samples of all devices, so as to make the training dataset become IID [7]. Wang et al. propose a method called P2PK-SMOTE, which is an improved version of Synthetic Minority Oversampling Technique (SMOTE), to generate more data points by this linear interpolation technique and make the dataset more IID [8]. Although the improvement of model's performance is significant by rebalancing the Non-IID dataset, this method may have some privacy issues since that the new data are generated by users' private data and shared with other users. Another type of solution is to modify the aggregation strategy. Onoszko et al. proposed an algorithm named Performance-based Neighbor Selection (PENS) to reduce the negative effect of Non-IID by choosing the most suitable neighbors' models to do the aggregation based on the test accuracy of them and as a result, fusion only happens between workers with more similar data distribution. But in our experiment scenarios, this algorithm may lead to a more severe overfitting problem in every worker due to the way Non-IID dataset is generated, so we modify this algorithm to get a better performance.

III. P2P Decentralized Federated Learning Simulation System with Ad-hoc like networks

A. Device Movement Simulation

The main purpose of our work is to perform decentralized federated learning in a dynamic network, where mobile devices are moving which results in a changing network topology. To achieve this, we use the famous Random Way Point (RWP) model to simulate the movement of devices. RWP model is a random model for the movement of mobile users, and how their location, velocity and acceleration change over time [6]. Every device starts by pausing for a fixed number of seconds, then selects a random destination in the simulation area and a random speed in the given range. It moves to its destination and pauses for another fixed period before it starts the next round of selecting destination and speed. In our federated learning scenario, the concept of time can be replaced by training epoch, one training epoch stands for one second in the original movement model. This doesn't mean that the training epoch really spends one second in real life, but only an assumption to test the system more conveniently.

B. Non-IID Dataset Generation

In order to simulate the unbalanced data distribution of mobile devices in real life, it's necessary to generate some Non-IID dataset. The authors of [3] generate a Non-IID dataset by rotating the image by some different angles to study about the covariate shift problem. However, the problem we focus on is the unbalanced classes in every device. Take the famous MNIST dataset as an example, which is a large dataset of handwritten digits from zero to ten. Suppose every device has the same size of dataset of 6000 images, a device can hold 5000 images of number "0" and 1000 randomly chosen images of other numbers, while another device holds 5000 images of number "1" and 1000 other numbers' images. In this way, the labels of data of each device are unbalanced.

C. Training Procedure

In our system, we have 10 workers, each $worker_i$ holding its own small biased train dataset $\mathcal{D}_i$, local model M_i with parameters W_i, an optimizer Op and a loss function $Loss$. Besides, to test the performance of every worker, all workers have a same balanced test dataset $\mathcal{T}$. As described in the RWP model, workers are set at a random location in a 1000x1000 area. They start by pausing for fixed epochs t, then choose a random destination within the area and a random speed from the range of $[v_{min}, v_{max}]$, pause for e epochs again after arriving at the destination and then start another round of movement. The location of $worker_i$ at epoch t is denoted by l_i^t. $worker_i$ can communicate with $worker_j$ if the euclidean distance euc_{ij} between them is less than a threshold d, and we call them neighbours. In each epoch t, we calculate the distance between every two workers and $worker_i$ will get a list of model parameters $\mathcal{N}_i^t$ by communicating with its neighbours. Specially, Wi is always in $\mathcal{N}_i^t$.

Workers communicate with their neighbours and do deep learning in a P2P way during the movement. There are two training strategies for every worker:

1) Only train when it has neighbours. In each epoch, if

there's only one element W_i in nbr_i^t, $worker_i$ will not train its local model and just keep moving to the next location. And if it has other neighbours, it will communicate with them and get their local model parameters, and use an aggregation function to calculate its new local model, then train this model for one epoch using its local dataset and optimizer function.

2) Always training. The difference between this strategy and the first one only happens when $worker_i$ has no neighbour in an epoch. Instead of stop training, it still trains the local model for one epoch.

D. Aggregation Function

When a worker communicates with others, it needs to use a specific aggregation function to decide how to make use of the model parameter information gathered from them. The aggregation function takes the model parameters list of a worker's neighbours at an epoch as input, and outputs the new local model. A commonly used aggregation function is a simple average function which is shown as

$$Avg(\mathcal{N}_i^t) = \frac{\sum_{W_k \in \mathcal{N}_i^t} W_k}{|\mathcal{N}_i^t|}. \quad (1)$$

However, if local model aggregates with a bad performing model using this algorithm, the performance will be negatively affected. Trying to reduce this bad effect, we let every worker stores a same but small test dataset whose classes are well balanced. Every time a worker trains its local model, it will calculate and save the newest model's test accuracy on the test dataset. When $worker_i$ sends W_i to neighbours, it also needs to send the test accuracy of local model acc_i^t to them. As a result, worker gets a list of neighbours' test accuracy $\mathcal{A}_i^t$. The weighted average is calculated during aggregation using the test accuracy as each model's weight instead of simple averaging, which is shown as

$$WeightedAvg(\mathcal{N}_i^t, \mathcal{A}_i^t) = \frac{\sum_{W_k \in \mathcal{N}_i^t} W_k * acc_k^t}{\sum_{W_k \in \mathcal{N}_i^t acc_k^t}}. \quad (2)$$

The whole training procedure with weighted average aggregation function is summarized in Algorithm 1, RWP_Move simulate the movement of workers, $OnlyNbr$ is a boolean value which is true when a worker trains its model only when it has neighbours other than itself, and $Send(x, j)$ means to send information about element x directly to $worker_j$. The purpose of the experiment is to test the computational model, so we didn't consider about the synchronization problem and assume that the information exchange can be finished in a moment.

Algorithm 1 P2P Federated Learning Algorithm with Ad-hoc Network Using Weighted Average Function

for Epoch t from 0 to ... **do**
  for Workers i from 0 to 9 **do**
    $\mathcal{N}_i^t \leftarrow \emptyset$
    $\mathcal{A}_i^t \leftarrow \emptyset$
    $l_i^t = RWP\_Move(l_i^{t-1})$
  end for
  for Workers i from 0 to 9 **do**
    for Workers j from 0 to 9 **do**
      $euc_{ij} = Calculate\_Distance(l_i^t, l_j^t)$
      if $euc_{ij} < d$ **then**
        $Send(W_i^{t-1}, j)$
        $Send(acc_i^{t-1}, j)$
        $\mathcal{N}_i^t.Append(W_j^{t-1})$
        $\mathcal{A}_i^t.Append(acc_j^{t-1})$
      end if
    end for
    if $|\mathcal{N}_i^t| > 1$ or $OnlyNbr == False$ **then**
      $W_i^t \leftarrow WeightedAvg(\mathcal{N}_i^t, \mathcal{A}_i^t)$
      $W_i^t \leftarrow Opt(W_i^t, \mathcal{D}_i)$
      $acc_i^t \leftarrow Test(W_i^t, \mathcal{T})$
    else
      Continue
    end if
  end for
end for

IV. Experimental Setup

We conducted experiments on a Linux Ubuntu server equipped with a 24G NVIDIA-TITAN-RTX GPU. The federated learning system is implemented by Python, and neural network models are built with the help of Pytorch.

A. Datasets

We separate two famous big image datasets, MNIST10 and CIFAR10, into several small Non-IID datasets. MNIST10 consists of 60,000 28x28 grayscale training images and 10,000 test images, while CIFAR10 contains 60,000 32x32 colored training images and 10,000 test images. Both of the datasets have 10 different classes to be categorized. With the method mentioned in Section III, we create *90MNIST* and *60CIFAR* datasets, where "90" and "60" stand for the percentage of major class in every small dataset. The larger this percentage is, the more Non-IID our dataset will become. Since we have ten workers, every worker holds approximately 6,000 images in *90MNIST* and 5000 images in *60CIFAR*. Table I shows the components of every worker's training dataset when using *60CIFAR*.

B. Models and hyperparameters

We implement two Convolutional Neural Networks (CNN) for image categorization tasks of two widely used datasets in Computer Vision (CV) field. The smaller CNN designed for *90MNIST* consists of 4 convolutional layers with a maximum dimension of 128 while the bigger one for *60CIFAR* contains

TABLE I: Components of every worker's dataset when using *60CIFAR*

Worker	airplane	automobile	bird	cat	deer	dog	frog	horse	ship	truck	**Summary**
0	**3194**	184	194	204	192	203	202	210	200	208	4991
1	186	**3299**	188	230	211	196	202	195	199	196	5102
2	203	191	**3174**	192	207	198	195	205	183	215	4953
3	209	195	184	**3179**	209	193	199	173	210	217	4968
4	195	177	215	200	**3206**	208	198	189	215	223	5026
5	195	191	205	205	193	**3220**	199	220	184	207	5019
6	224	205	216	195	195	195	**3210**	193	213	187	5033
7	196	185	204	187	179	202	212	**3201**	228	201	4995
8	189	195	201	224	195	181	200	208	**3183**	172	4948
9	209	188	219	184	213	204	183	206	185	**3174**	4965
Summary	5000	5000	5000	5000	5000	5000	5000	5000	5000	5000	50000

6 convolutional layers among which the maximum dimension is 256. Between convolutional layers there are ReLu as activation function, max-pooling or average-pooling function for downsampling, dropout layers to deal with the severe overfitting problem that might happens when using Non-IID datasets, and batch normalization layers. Outputs of the final convolutional layer will be fed into a fully connected layer to get the final outputs of the whole network. We didn't use some state-of-art models like Vgg-16 [9] and Densenet [10] because they are too big and slow to train for mobile devices whose hardware resources are limited. We choose Adam as our adaptive optimizer whose learning rate is set to $1e-3$, and cross-entropy loss as our loss function.

For RWP model, the speed range of every worker is set to $[3, 7]$, the communication distance threshold $d = 100$, and the fixed pausing epochs $t = 10$.

C. Training Scenarios and Baselines

There are three baselines for both two datasets,

1) `Central`: Train a single model using all of the dataset non-distributed in one worker. This baseline reflects the upper limit of model's performance on the given dataset.
2) `Self`: Workers only train with their local small datasets without communicating with other workers.
3) `Fully`: In this baseline, instead of a dynamic ad-hoc like network topology, the network in which workers are placed is fully connected.

And there are also several different scenarios for training,

1) `OnlyNbr`: As mentioned in Section III, workers stop training local model if there isn't any neighbour worker to communicate with. They can only train in a P2P way when they have neighbours.
2) `NotOnly`: Workers still train themselves even if they don't have any neighbour. The first two scenarios both use ordinary average function as shown in (1) as their aggregation function without considering about the weights.

TABLE II: Average maximum accuracy of workers and maximum accuracy of global model after 2000 epochs with *90MNIST* and *60CIFAR* (denoted by M and C)

Scenario	**M_Avg(%)**	**M_Global**	**C_Avg**	**C_Global**
`Central`	99.59	-	84.21	-
`Self`	90.48	12.48	41.30	12.80
`Fully`	99.44	99.45	71.96	74.53
`OnlyNbr-W`	98.90	99.46	60.34	74.69
`NotOnly-W`	99.06	99.50	59.76	74.62

3) `OnlyNbr-W, NotOnly-W`: These two scenarios just change the aggregation functions in the first two scenarios into the $WeightedAvg$ mentioned in (2).
4) `OnlyNbr-M`: This scenario uses a mixed training strategy for `OnlyNbr`. In the first 800 epochs, workers use ordinary average function as their aggregation function, and switch to $WeightedAvg$ for the remaining epochs.

D. Evaluation Method

In order to test the performance of models for all scenarios and use $WeightedAvg$ function, every worker also needs to download the same test dataset which is balanced in different labels while they get initial models. In every epoch, each worker calculates the test accuracy of the local model on the test dataset. There are two evaluation metrics for the performance of whole learning system in a specific epoch:

- **Global Model Accuracy.** We generate a "global model" by applying weighted average function to all workers' models every epoch and measure test accuracy of this "global model" on the test dataset. The best accuracy of this global model can reflect the best performance of the whole system during the entire training procedure.
- **Average Maximum Accuracy.** We keep track of workers' test accuracy and record the accuracy history for them. In each epoch, we calculate the average value of all workers' maximum test accuracy until this epoch, which reflects the average best performance of workers.

TABLE III: Average maximum accuracy for workers after 100∼2000 epochs with *60CIFAR*

Scenario	100_Avg(%)	500_Avg	1000_Avg	2000_Avg
Central	84.21	84.21	84.21	84.21
Self	41.30	41.30	41.30	41.30
Fully	53.77	68.66	71.96	71.96
OnlyNbr-W	19.73	51.81	55.59	60.34
OnlyNbr-M	19.43	51.54	55.76	61.78
OnlyNbr	16.85	46.21	54.17	61.40
NotOnly-W	36.49	46.35	52.72	59.76
NotOnly	36.82	46.72	51.86	59.51

TABLE IV: Maximum global model's accuracy for workers after 100∼2000 epochs with *60CIFAR*

Scenario	100_Glob(%)	500_Glob	1000_Glob	2000_Glob
Self	12.80	12.80	12.80	12.80
Fully	73.51	74.53	74.53	74.53
OnlyNbr-W	50.68	71.04	72.47	73.96
OnlyNbr-M	50.36	71.33	73.35	74.46
OnlyNbr	23.98	67.02	72.93	74.72
NotOnly-W	22.07	65.96	71.59	74.62
NotOnly	16.25	64.10	70.98	74.07

V. Analysis and Discussion

A. Comparison of Different Datasets

Table II shows the two evaluation metrics mentioned in the last section after training for 2000 epochs with both two datasets. The accuracy of `Central` baseline shows that CIFAR10 is much harder to train than MNIST10 with our CNN models. The results of `Self` baseline reflects the negative effects of workers' local small and unbalanced datasets. This effect is more obvious on harder learning dataset like CIFAR10, which lets accuracy to drop by more than 50%, while it only drops by about 10% with an easier dataset like MNIST10 even if *90MNIST* is much more "Non-IID" than *60CIFAR*. The differences between the results of `Fully` baseline and two proposed scenarios indicate the impact of network topology. When training with MNIST, the accuracy of workers in Ad-hoc wireless network is close to fully connected network if we let workers communicate with others and train in a peer-to-peer way. However, this accuracy difference is bigger if we use a harder CIFAR dataset, showing that network topology can have a great impact on model's performance of workers. But we can still improve workers' accuracy by 20% comparing to `Self` baseline, which means that we can reduce the accuracy drop caused by an unbalanced dataset with P2P communication.

Since all scenarios except for `Self` can have a similarly good performance when testing with *90MNIST* and it's hard to tell the difference between scenarios using this kind of easy dataset, we mainly consider about the experiment results using *60CIFAR* for the rest of analysis.

B. Comparison of Different Communication Strategies

Table III and IV show different scenarios' test results of two evaluation metrics after different training epochs. From the data, we can notice that the strategy `OnlyNbr` where we only train workers when they have neighbours is better than `NotOnly` where workers keep training themselves no matter what aggregation function we choose or what learning metrics we select in most cases. The latter strategy is only better in "100_Avg" case from Table III, and it's because workers don't have many chances to communicate with other workers and train their local models.

Moreover, the real training rounds of our system using `OnlyNbr` strategy are much fewer than using `NotOnly`. It's resonable because when workers train only with their unbalanced local dataset without communication, it's more possible for overfitting problem to occur. As a conclusion, `OnlyNbr` is not only better in model performance, but also much less resources consuming, which is important for IoT devices whose resources are highly limited.

C. Discussion about System Performance and Application

If the system is trained with Ad-hoc like wireless networks in P2P way, we can still get a well performed global model even though the test accuracy of every worker is low, as shown in last two scenarios in Fig. 1. In fact, the average maximum accuracy of `OnlyNbr` keeps improving and is approximately 70% after 5000 training epochs, while the maximum global model accuracy stays around 75%, which is not quite different from 2000 epochs. This may infer that when we use dynamic network topology, the situation is similar to fully connected networks that the whole system converges and the best performance of worker's model gets close to the global one in the end, though this procedure can be much slower than `Fully`. In other words, the network topology mainly influences the converge speed of our learning system, but not the best performance.

As a consequence, local models' performance will never get better than global model and improve much slower than global model with hard dataset like *60CIFAR* and non-fully-connected network topology. Since that, we need to make full use of this global model if possible. We can get a fairly satisfying global model after only 500 training epochs in our scenarios, and it gets better much slower after that. So for application, workers can upload their local model and generate a global model after first few epochs, and train the new global model in a pure P2P way for another round.

It's also possible to use workers' local models for application directly without a data center, but it takes too many training rounds for them to become as good as the global model and this is hard to achieve in real-life scenarios. However, if we use easier datasets like *90MNIST* or better network topology, this strategy might be applicable.

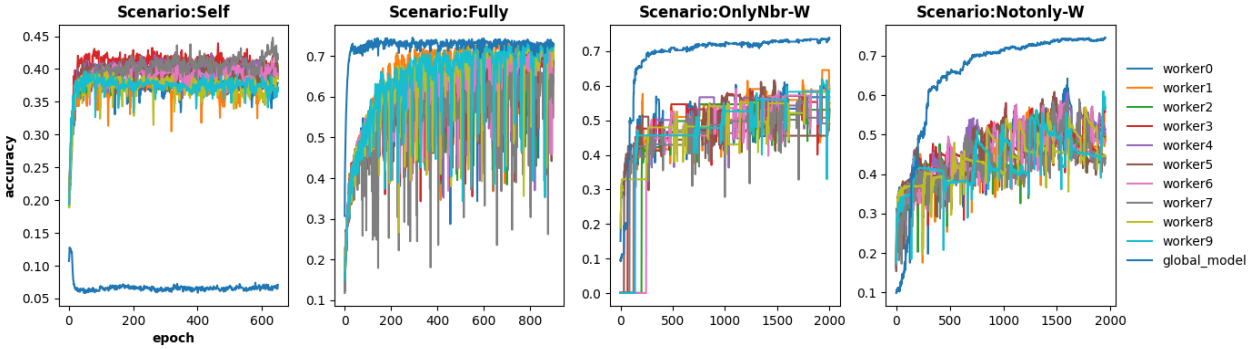

Fig. 1: Comparison of four scenarios' test accuracy with *60CIFAR* dataset, using weighted average as aggregation function.

D. Comparison of Different Aggregation Functions

Since we have already got the conclusion that `OnlyNbr` is a better strategy, we only discuss the impact of different aggregation functions can have on this strategy.

Fig. 2 shows the performance of the whole system when using different aggregation functions for `OnlyNbr` communication strategy in the first 5000 training epochs. The accuracy improves faster when using weighted average function as aggregation function than non-weight average function at first in both metrics. This is because when two models merge, the better one will be negatively effected by the worse one, and there are many workers that have only trained few rounds and perform badly in `OnlyNbr` strategy, which should be given a smaller weight when fusing. However, the system with non-weight average function will outperform weighted version slightly in the end. And the mixed aggregation function selection strategy, which uses weighted average in the beginning of training procedure and switches to non-weight version later, combines the advantages of two functions and is the best strategy among these three.

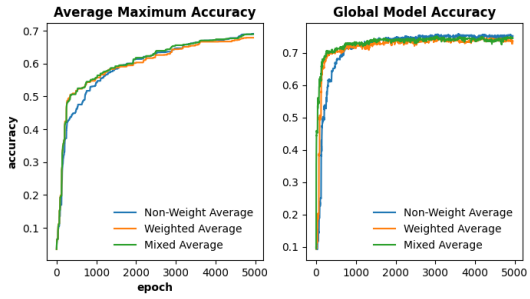

Fig. 2: Comparison of two evaluation metrics with *60CIFAR* dataset and three different aggregation functions using `OnlyNbr` strategy for 5000 epochs

VI. Conclusions

In this work we implement a decentralized P2P federated learning system, tested with two Non-IID datasets and Ad-hoc like dynamic network topology. We also try different P2P communication strategies and aggregation functions. The experiment results show that for communication strategy, it's better to let worker train their models in a P2P way only when they have neighbours to communicate with than keep training themselves. We also design an efficient aggregation function selection strategy by mixing the non-weighted average function and weighted average function. And after 5000 training epochs, the average maximum accuracy of workers of `OnlyNbr-M` scenario reaches 69%, with 28% improvement from `Self` scenario, which indicates that P2P communication can reduce the negative effects caused by unbalanced datasets significantly. From the comparison between the testing results of different network topology, we find that topology can only influence the converge speed of whole system's best performance.

Besides, there are many possible directions of improvement. For future work, we can try to scale up the whole system and use more workers to do the experiments. This can be hard to implement because the training speed is already slow by only using 10 workers. It's all about the trade-off between speed and accuracy. We can either find a dataset which is easier to train than CIFAR10 but harder to train than MNIST10, or use a simpler model for training. Furthermore, many researchers have tried different methods to deal with the Non-IID problems. Which kind of methods can be applied to our system is worth trying as well. And for now, the design of our aggregation functions is still naive in some degree. We'll try to find or design a best aggregation strategy which is most suitable to our system in the future.

References

[1] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Artificial intelligence and statistics*, pp. 1273–1282, PMLR, 2017.

[2] S. Basagni, M. Conti, S. Giordano, and I. Stojmenovic, *Mobile ad hoc networking*. John Wiley & Sons, 2004.

[3] N. Onoszko, G. Karlsson, O. Mogren, and E. L. Zec, "Decentralized federated learning of deep neural networks on non-iid data," *arXiv preprint arXiv:2107.08517*, 2021.

[4] C. Pappas, D. Chatzopoulos, S. Lalis, and M. Vavalis, "Ipls: A framework for decentralized federated learning," in *2021 IFIP Networking Conference (IFIP Networking)*, pp. 1–6, IEEE, 2021.
[5] J. Benet, "Ipfs-content addressed, versioned, p2p file system," *arXiv preprint arXiv:1407.3561*, 2014.
[6] S. Mao, "Fundamentals of communication networks," in *Cognitive Radio Communications and Networks*, pp. 201–234, Elsevier, 2010.
[7] E. Jeong, S. Oh, H. Kim, J. Park, M. Bennis, and S.-L. Kim, "Communication-efficient on-device machine learning: Federated distillation and augmentation under non-iid private data," *arXiv preprint arXiv:1811.11479*, 2018.
[8] H. Wang, L. Muñoz-González, D. Eklund, and S. Raza, "Non-iid data re-balancing at iot edge with peer-to-peer federated learning for anomaly detection," in *Proceedings of the 14th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, pp. 153–163, 2021.
[9] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.
[10] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4700–4708, 2017.