Robustness Analytics to Data Heterogeneity in Edge Computing

Jia Qian[a], Lars Kai Hansen[a], Xenofon Fafoutis[a], Prayag Tiwari[b], Hari Mohan Pandey[c]

[a]*Department of Applied Mathematics and Computer Science, Technical University of Denmark, 2800 Lyngby, Denmark.*
[b]*Department of Information Engineering, University of Padova, Italy.*
[c]*Department of Computer Science, Edge Hill University, Ormskirk, United Kingdom.*

Abstract

Federated Learning is a framework that jointly trains a model *with* complete knowledge on a remotely placed centralized server, but *without* the requirement of accessing the data stored in distributed machines. Some work assumes that the data generated from edge devices are identically and independently sampled from a common population distribution. However, such ideal sampling may not be realistic in many contexts. Also, models based on intrinsic agency, such as active sampling schemes, may lead to highly biased sampling. So an imminent question is how robust Federated Learning is to biased sampling? In this work[1], we experimentally investigate two such scenarios. First, we study a centralized classifier aggregated from a collection of local classifiers trained with data having categorical heterogeneity. Second, we study a classifier aggregated from a collection of local classifiers trained by data through active sampling at the edge. We present evidence in both scenarios that Federated Learning is robust to data heterogeneity when local training iterations and communication frequency are appropriately chosen.

Keywords: Intelligent Edge Computing, Fog Computing, Active Learning, Federated Learning, Distributed Machine Learning, User Data Privacy

1. Introduction

Federated Learning [1] is a promising method to enable edge Intelligence and data protection at the same time. FL is of significant theoretical and practical interest. From a theoretical point of view, Federated Learning poses challenges in terms of, e.g., consistency (do distributed learning lead to the same result as centralized learning) and complexity (how much of the potential parallelism gain is realized). From a practical point of view, Federated Learning offers unique opportunities for data protection. In particular, Federated Learning can be realized without "touching" the training data, but rather the data remains in its generation location, which provides the opportunity to secure user privacy. It is very intrinsic to bring it to IoT

Email addresses: `jiaq@dtu.dk` (Jia Qian), `lkai@dtu.dk` (Lars Kai Hansen), `xefa@dtu.dk` (Xenofon Fafoutis), `prayag.tiwari@dei.unipd.it` (Prayag Tiwari), `pandeyh@edgehill.ac.uk` (Hari Mohan Pandey)

[1]`https://github.com/jiaqian/robustness_of_FL`

application, in particular, when 5G is arriving. For instance, FL is exerted in industrial IoT (IIoT) to predict electric drivers' maintenance in the fog computing platform [2]. The medical data collected from distributed individuals can be processed locally and share the metadata with the central server at some point to protect personal privacy [3]. Extending FL to other machine learning paradigms, including reinforcement learning, semi-supervised and unsupervised learning, active learning, and online learning [4, 5] all present interesting and open challenges. Some works assume that data is **Independent** and **Identically Distributed** (IID) on the edge devices, which is evidently a strong assumption, say in a privacy-focused application. Users are not identical; hence, we expect locally generated dataset to be the result of idiosyncratic sampling, namely, biased. We believe that data diversity is not necessarily harmful in terms of performance, which mainly attributes to the aggregation step of FL, with the condition that local training iterations and batch size are appropriately opting. A high-level depiction of this scenario is presented in Figure 1.

To investigate the robustness of FL, we consider two types of Non-IID cases: Type i we will simulate a highly biased data-generation environment, edge devices have access only to a subset of the classification classes (no overlap between them); Type ii on the edge devices, we employ AL as an active sampler to sample the most representative instances, rather than uniform sampling.

1.1. Contribution

Our contribution can be summarized as:

- In general, we aim to investigate the relationship between distributed data diversity and centralized server performance in the edge computing environment.
- More specifically, we simulate two types of biased data generation to study the robustness of FL to different unbalanced data generation level.
- Our experiments show that centralized server performance is highly correlated to the local training time and communication frequency. The divergent aggregation might happen if they are not appropriately chosen.
- Finally, we investigate the effects of parameter (gradient) aggregation by comparing local neural networks activation patterns and aggregated neural networks, which shows the evidence that the server's classification capability is "inherited" from distributed devices through aggregation.

1.2. Organization

The remainder of this paper is organized as follows: Section 2 we will explain the preliminary concepts and introduce the related work, in Section 3, we will give the specific introduction of our scheme. In Section 4 the details of our experimental results will be recovered. Section 5 we will conclude the paper.

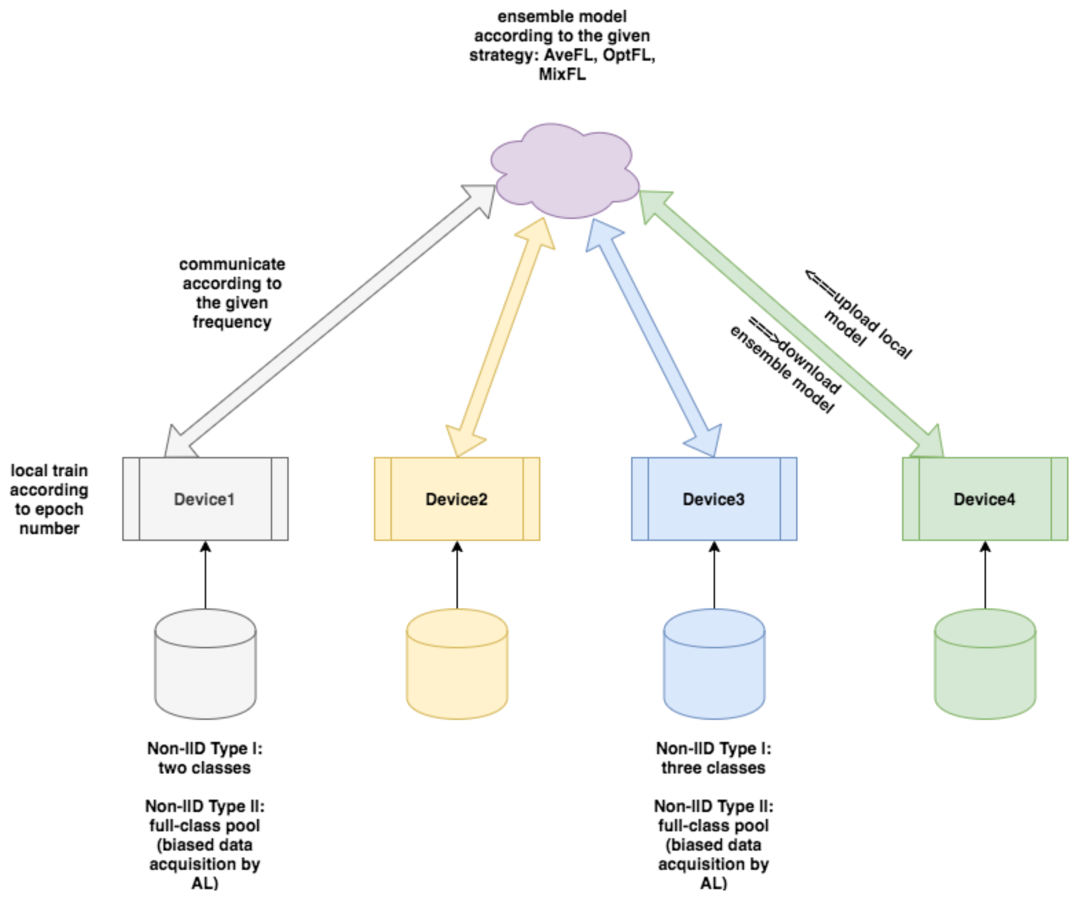

ensemble model according to the given strategy: AveFL, OptFL, MixFL
communicate according to the given frequency
<==upload local model
==>download ensemble model
local train according to epoch number
Device1
Device2
Device3
Device4
Non-IID Type I: two classes
Non-IID Type II: full-class pool (biased data acquisition by AL)
Non-IID Type I: three classes
Non-IID Type II: full-class pool (biased data acquisition by AL)

Figure 1: Federated Learning Scheme.

For the convenience of readers, we list all the abbreviations and annotations.

2. Preliminaries and Related Work

2.1. Federated Learning

FL uses a coordinated fashion to train a global model by dynamically collecting models from distributed devices for some rounds. It was first proposed by [1] for the user privacy consideration in mobile networks, and it is a very practical framework in edge computing. [6] employs FL to detect attacks in a distributed system, [7] predicts model uncertainty by a deep aggregated model, and [8] aims to optimize the structure of neural network in FL. Some FL-based applications assume the data is IID on edge devices. [9] considers Non-IID data, but it focuses on the observation that accuracy reduction caused by Non-IID is correlated to weight diversity. Our work extends it, studying two types of Non-IID data: (i) Type i we will simulate a highly biased data-generation environment, whereby edge devices can only generate their categories without

Table 1: Abbreviations & Annotations

Abbreviations	Full Name
FL	Federated learning
Al	Active Learning
IID	Independent Identical Distributed
AveFL	Average Federated Learning
OptFL	Optimal Federated Learning
MixFL	Mixed Federated Learning
Mc-drop	Monte carlo dropout
Non-IID Type i	Active Learning sampler
Non-IID Type ii	Non overlap between categories

any overlap between devices, (ii) Type ii whereby we employ AL on the edge devices as an active sampler to simulate a slightly biased data generation.

2.2. Active Learning of Neural Networks

Labeling is challenging and expensive when data generation increases exponentially. Thus, when intelligence sits close to edge users, it is therefore natural to utilize the interaction between machines and users/humans. We combine Federated Learning (FL) and Active Learning (AL) as Non-IID Type ii, and we reported the prototype in [10]. Theoretically, AL may achieve one of the following situations: higher accuracy with the same amount of data or with a given performance using fewer data. According to the formula of incoming data, it can be grouped as pool-based and stream-based. The stream-based AL approach is used when the data arrives in a stream way, and the model must decide whether to query from the "Oracle" or discard it.

The pool-based approach (Figure 2) is composed of an initially trained model, an "Oracle", an unlabeled data pool, and a small labeled dataset. More specifically, the initially trained model elaborately opts for some representative samples out from the unlabeled pool based on the acquisition function. After that, it asks the oracle to label them and includes the labeled ones to the training set for future training. We can repeat such operations for several times. In the previous work [10, 11], we train our model whenever lately-labeled data is added, along with the old (labeled) data. In this paper, we consider Online Active Learning, which means we immediately discard the data after training (more details in Section 3.3), but with a negligible small subset shared across the devices. To apply AL on a neural network, we firstly build a Bayesian Neural Network (BNN), which can be considered as a model that outputs different values for the same input fed in the model several times. There is no analytical form of the posterior distribution in the neural network;

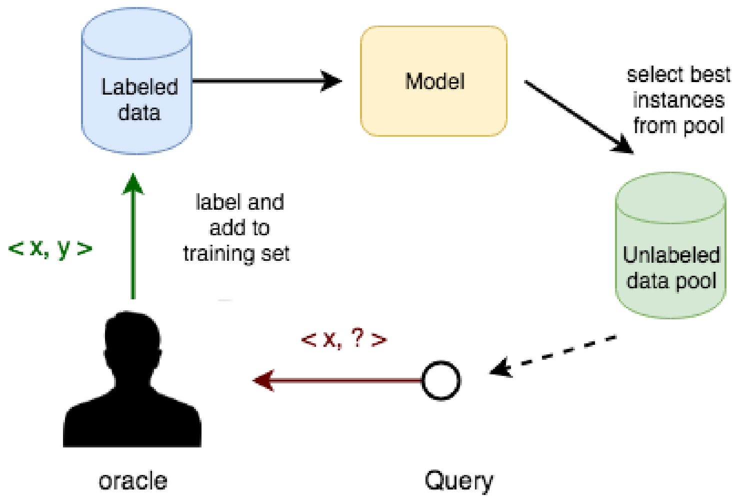


Figure 2: Pool-based Active Learning Scheme.

typically a surrogate distribution q is introduced to approximate it by minimizing the distance between them. We implement it through a free ride Dropout [12], feeding the same input multiple times to approximate a distribution with certain mean and variance. It can be proved that running dropout is approximated to apply Bernoulli prior on the model parameters. More details can be found in [10].

2.3. Boosting Approach

Boosting is designed to improve any machine learning method, e.g., tree-like classifiers, by aggregating many weak learners through bias and variance reduction [13] [14]. The approach of the present work can be related to boosting by viewing the aggregation as a combination of 'weak' (specialized) edge models in repeated steps of the federation. In [15], the authors proposed the Boosting Gradient Classifier, which has a set of weak learners and sets off by creating a weak learner, and it keeps increasing after every iteration. The set of learners is built by randomly combining features. It seeks an appropriate combination $\hat{F}$ of f_i such that approximates the true F, expressed as $\hat{F}(x) = \sum_{i=1} \beta_i f_i(x)$. Apart from computing gradients during training, it also computes the second-order derivative to decide the learning rate. Instead, our method keeps the number of weak learners constant, which is the number of edge devices. Analogously, we can also make it dynamic, like boosting gradient classifiers. Another difference is that we do not compute the second-order derivatives to decide the learning rate; instead, we empirically choose one as the Neural Network has a large amount of parameters. The Boosting Gradient classifier typically has a good performance in conventional machine learning application [16, 17, 18]. The main steps of Boosting Gradient are as follows:

- if $m = 0$, we output the prediction by average the outcomes from the weak learners $\hat{F}(x) = \frac{1}{n}\sum_{i=1}^{n} y_i$.
- For iteration m from 1 to M (the case $m \neq 0$):

– model in iteration m defines as

$$W_m = W_{m-1} + \gamma_m g_m(x)$$

– γ_m and $g_m(.)$ are computed separately by the first and second order of loss function L.

$$g_m(x) = -[\frac{\partial L(y_i, \hat{F}(x, W_m))}{\partial W_m}]_{W=W_{m-1}}$$

$$\gamma_m = [\frac{\partial^2 L(y_i, \hat{F}(x_i, W_m))}{\partial W_m^2}]_{W=W_{m-1}}$$

F_{m-1}: the collection of learners up to stage m-1.

γ_m: the learning rate in iteration m.

$g_m(x)$: gradient in stage m.

In summary, both Boosting Gradient classifier and FL attempt to improve the performance by assembling a set of weak models. The Boosting Gradient classifier works on a dataset with extracted features, specifically, optimize the learning rate and keeps the number of weak learners increasing; whereas we design federated learning for neural network, the learning rate is empirically decided due to the computation problem and the size of models is constant, we can make it dynamic though.

2.4. Other works

Data non-IID was introduced in [9], and they tackle it by introducing a relatively small global subset that may somehow capture the whole distribution, shared across all devices. Similarly, [19] suggests using data distillation to extract a low-dimension (or sparse) representation of the original data. However, it is computationally expensive; in particular, it is typically carried out at the edge side where only little computation resources can be offered. [20] converts non-IID data distribution as an advantage by considering it as a multi-task optimization, which conforms to our conclusion. Furthermore, [21] utilizes distributionally robust optimization to minimize the worst-case risk over all the distributions close to the empirical distribution.

3. Proposed scheme

3.1. Federated Learning Aggregation Strategies

More specifically, let's assume the server shares the model (at round t) W_t with n devices for their local updating, and the updated models are denoted as $W_t^1, W_t^2, W_t^3, ...W_t^n$. Then, the devices upload the improved models to the server, and the server outputs the aggregated model according to the following criterion:

$$W_{t+1} := \sum_{i}^{n} \alpha_i * W_t^i \qquad (1)$$

Algorithm 1 *AveFL*

```
1: Input: $W_j^t$: local models at round $t$
2: Output: aggregated model $W^t$
3: $W^t = \frac{1}{n}\sum_{j=1}^{n} W_j^t$
```

Algorithm 2 *OptFL*

```
1: Input: $W_j^t$: local models at round $t$, $A(.)$: measure accuracy
2: X: test dataset
3: Output: aggregated model $W^t$
4: $W^t = \text{argmax}\, A(\text{BNN}(X, W_j^t))$ for $j = 1, 2, .., n$
```

The combination weights α_is can be uniformly distributed or determined to reflect network performance. The former is referred to as *AveFL* (Algorithm 1). The learning process is iterative. We also consider the second scheme, where we opt for the highest-accuracy model, namely, set α^* of the best model equal to one, and the rest to zero, labeled as *OptFL* (Algorithm 2). In Section 4, we evaluate the schemes and a combination of *AveFL* and *OptFL*, named as *MixFL*. The latter selects the best model of the former two (Algorithm 3).

Rather than aggregating the weights of models in Equation (1), we can also work on the gradients. We conclude that one-batch weight average is equal to gradient average. Suppose we have n devices, and training data D ($|D| = N$) is sectioned into n parts as $D_1, D_2, ...D_n$, $|D_1| = N_1, |D_2| = N_2, ...|D_n| = N_n$. The corresponding weights inferred from D_i is W_i. Then we define a cost function $G(D) = \sum_{i=1}^{N} g(\widetilde{y}_i, y_i, w)$ and the initial model is W_0, β is the learning rate. We first define average one-batch weights of models as shown in Equation (3), notably, the local update of edge devices is after one batch (no iteration of the batch), otherwise it is **not** W_0 in cost function $g(.)$. The gradient aggregation is defined in Equation (4).

$$\sum_{i=1}^{n} \alpha_i = 1 \tag{2}$$

Aggregation Weights:

Algorithm 3 *MixFL*

```
1:  Input: local models at round $t$ $W_j^t$
2:  Output: aggregated model $W^t$
3:  $W_{\text{ave}}^t = AveFL(W_j^t)$
4:  $W_{\text{opt}}^t = OptFL(W_j^t)$
5:  $\text{acc}_{\text{ave}} = A(\text{BNN}(X, W_{\text{ave}}^t))$
6:  $\text{acc}_{\text{opt}} = A(\text{BNN}(X, W_{\text{opt}}^t))$
7:  if $\text{acc}_{ave} >= \text{acc}_{opt}$ then
8:      Return $W_{\text{ave}}^t$
9:  else
10:     Return $W_{\text{opt}}^t$
```

$$\begin{aligned} W &:= \sum_{i=1}^{n} \alpha_i (W_0 + \beta G(D_i)) \\ &= \sum_{i=1}^{n} \alpha_i \underbrace{(W_0 + \beta \frac{1}{N_i} \sum_{j=1}^{N_i} g(\widetilde{y}_j, y_j, \mathbf{W_0}))}_{\text{updated W from device i}} \\ &= W_0 \sum_{i=1}^{n} \alpha_i + \beta \sum_{i=1}^{n} \alpha_i \frac{1}{N_i} \sum_{j=1}^{N_i} g(\widetilde{y}_j, y_j, W_0) \\ &= W_0 + \beta \sum_{i=1}^{n} \alpha_i \frac{1}{N_i} \sum_{j=1}^{N_i} g(\widetilde{y}_j, y_j, W_0) \end{aligned} \tag{3}$$

Aggregation Gradients:

$$\begin{aligned} W &:= W_0 + \beta * (\sum_{i=1}^{n} \alpha_i G(D_i)) \\ &= W_0 + \beta (\sum_{i=1}^{n} \alpha_i \underbrace{\frac{1}{N_i} \sum_{j=1}^{N_i} g(\widetilde{y}_j, y_j, W_0))}_{\text{gradient of device i}} \end{aligned} \tag{4}$$

In each iteration, keeping W_0 the same for all edge devices is a mandatory step; otherwise, weight divergence can occur. If initial models are different, they might be placed in a different low-cost region of the cost landscape. Thus, after the average aggregation step, it might be sub-optimal. In this paper, we also aim to investigate how the number of local training influences the result, and we decide to work on the weights aggregation for the sake of convenience.

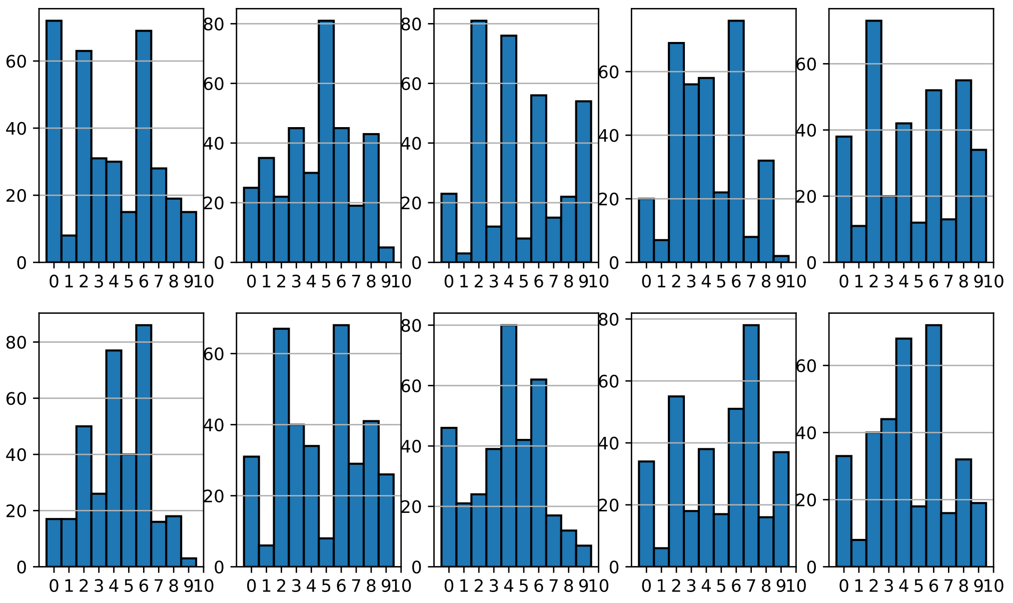


Figure 3: Biased Data Acquisition by Active Learning: we demonstrate the distribution of data acquired by AL for 10 acquisitions. They are unbalanced in different ways for every acquisition.

3.2. Method for Non-IID Type i

Our approach can be divided into two stages: local learning and aggregation. The two stages will be iterated in one round.

1. **Initialization:** At initialization, the centralized server trains an initial model W^0 with m data samples. More general, we define the model as W^t, where t indicates the current round number.
2. **Sharing:** Server shares the model W^t with n activated edge devices $d_1, d_2, ..., d_n$.
3. **Local Training:** All edge devices implement local training and update their models $W_1^t, W_2^t, ..., W_n^t$. This step incorporates one or multiple cycles of data acquisition.
4. **Aggregation:** Edge devices transmit their corresponding models to server and the server aggregates $W_i^t, i = 1, 2.., n$ to get W^{t+1}. The aggregation could be *AveFL*, *OptFL* or *MixFL*.
5. Repeat steps 2-4 if necessary.

Algorithmically, it is described in Algorithm 4.

3.3. Method for Non-IID Type ii

We consider FL with AL as Non-IID Type ii since AL samples a subset of data with higher uncertainty, which leads to biased samples. First, we divide the whole training set into four parts, one part for one edge device. Then we build a pool with 4000 images randomly sampled from one part for the computation

Algorithm 4 Non-IID Type i

```
1:  Input: $X_1 \cup X_2 \cup X_3, .. \cup X_n,\ X_{\text{ini}}, W^0$
2:  Return: model $f(W)$
3:  if t==0 then
4:      $W^1 \leftarrow W^0 - \alpha \nabla f(X_{\text{ini}}; W^0)$
5:  else
6:      for t=1,2,..T do
7:          =>Devices:
8:          for j=1,2,..,n do n devices (in Parallel)
9:              random sample from $X_j$: $D_j^t \sim X_j$
10:             $\mathrm{W}_j^{t+1} = W_j^t - \alpha \times \nabla f(D_j^t; W_j^t)$
11:         end
12:         =>Server:
13:         Aggregation: $W^{t+1} = AveFL(\mathrm{W}_j^{t+1})$ for j=1,2,...,n
        end
14: end
```

consideration since we need to measure the uncertainty of every data point in the pool. The pool is almost balanced; however, the batches generated by the active sampler is unbalanced, one example of 10 acquisitions shown in Figure 3. For scalability, in this paper, we perform an online AL. Namely, the model is further trained only by the new batch, without the access of the old data (except small subset with 50 images), which is different from the previous work that we train all the data from scratch whenever new data is coming. We try to alleviate the forgetting problem of online learning by a cheap trick, storing 50 images, a balanced set (5 images per class) and will be combined with a new batch to train the model. After completing current-round training, we dump the new batch and only keep 50 images in the labeled set. Moreover, we also use weight decay [22] as a regularizer that prevents the model from changing too much. We define it in Equation 5, $E(.)$ is the cost function, w^t is the model parameter at round t and λ is a parameter governing how strongly large weights are penalized.

$$E(w^{t+1}) = E(w^t) + \frac{1}{2}\lambda \sum_i (w_i^t)^2 \tag{5}$$

[23] learns the weights that can mostly approximate the distribution of the data from the pool by solving an optimization problem. It is highly computation-demanding and not suitable for edge devices. Another work [24] attempts to avoid forgetting by dividing the NN architecture into parts and assigning them to different edge devices, but it requires restricting synchronization during aggregation. Our Non-IID Type ii method is sketched as:

Algorithm 5 Non-IID Type ii

```
1: Input: $X_1 \cup X_2 \cup X_3, .. \cup X_n,\ X_{\text{ini}}, W^0, k$
2: Return: model $f(W)$
3: if t==0 then
4:     $W^1 \leftarrow W^0 - \alpha \nabla f(X_{\text{ini}}; W^0)$
5: else
6:     for t=1,2,..T do
7:         =>Devices:
8:         for j=1,2,...,n do n devices (in Parallel)
9:             $\log p_j, p_j = \text{BNN}(f_j(W^t), x_j)$
10:            compute entropy: $S_j = -p_j \times \log p_j$
11:            sort in descending order and pick top k: $D_j^t = \text{sort}(S_j)[k]$
12:            $W_j^{t+1} = W_j^t - \alpha \times \nabla f(D_j^t; W_j^t)$
13:        end
14:        =>Server:
15:        Aggregation: $W^{t+1} = AveFL(\text{W}_j^{t+1})$ for j=1,2,...,n
16:    end
17: end
```

1. **Initialization:** In the beginning, a centralized server trains an initial model W^0 using m data samples. Without the loss of generality, we denote the model by W^t, where t is the current round.
2. **Sharing:** The central server shares the model W^t to n activated edge devices $d_1, d_2, ..., d_n$.
3. **Local Training:** All edge devices implement AL on a Bayesian Neural Network approximated by Dropout [12], locally train and update their models $W_1^t, W_2^t, ..., W_n^t$. This step incorporates one or multiple cycles of data acquisition.
4. **Aggregation:** Edge devices transmit their corresponding models to server and the server aggregates $W_i^t, i = 1, 2.., n$ to get W^{t+1}. The aggregation step could entail the average, performance-based or mixed mechanisms.
5. Repeat steps 2-4 if necessary.

The specific algorithm is described in Algorithm 5.

3.4. Architecture

Our model consists of four convolutional layers, one fully-connected layer and a softmax layer shown in Table 2. Note that we did not use batch normalization [25] in the architecture since the biased batch

Algorithm 6 *Bayesian Neural network* (BNN)

```
1: Input: $f_i(W^t), x_i$
2: Return: $\log \overline{p}, p$
3: $s = 0$
4: for g = 1,2,..r do
5:     $p = f_i(x_i; W^t)$
6:     $s += p$
7: end
8: $\overline{p} = \frac{1}{r} \times s$
```

normalization has a deleterious effect on the aggregation performance. Mathematically, it defines as shown in Equation 6 and Equation 7. Suppose we have a batch $B = \{x_j\}_{j=1,2..,m}$, then it is normalized by its mean μ_B and variance σ_B (computed in Equation (6)), and then we infer a new mean (β) and variance (γ) during training process. It may reduce the internal covariate shift and speed up the training procedure to form new representation of data (Equation (7)).

In the Non-IID case, the means (β) and variances (γ) optimized in the local training stage are decided by their biased data, and it is not beneficial during the aggregation stage in our experiments. It is very critical to enable aggregation effect in highly biased data generation; otherwise, the aggregated model performs very poorly (e.g., 20% accuracy with batch normalization and 47% otherwise).

$$\mu_B = \frac{1}{m}\sum_{i=1}^{m} x_i, \sigma_B^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2 \tag{6}$$

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \eta}}, y_i = \gamma \hat{x}_i + \beta \tag{7}$$

4. Experimental Results

4.1. Real Dataset

Fashion-MNIST (shown in Figure 4) is one benchmark image dataset published by Zalando, as the alternative of MNIST dataset. It is formed by a training set with 60,000 examples and a test set with 10,000 examples and 10 classes. One image has 28×28 pixels for width and height and one channel. Each pixel value ranges between 0 and 255, indicating the shades of grey.

4.2. Non-IID Type i

We first evaluate the case of Non-IID Type i: we have a ten-classes dataset and four edge devices (D1, D2, D3, and D4), randomly assign two classes to two devices and three classes to another two devices without

Table 2: Neural Network Architecture

layer	**layer name**	output channels or number of nodes	**kernel size**
1	**Conv2d**	64	4x4
2	**ReLu**	-	-
3	**Conv2d**	16	5x5
4	**ReLu**	-	-
5	**Max Pooling**	-	2x2
6	**Dropout**	-	0.25
7	**conv2d**	32	4x4
8	**ReLu**	-	-
9	**conv2d**	16	4x4
10	**ReLu**	-	-
11	**Max Pooling**	-	2x2
12	**Dropout**	-	0.25
13	**Linear**	128	-
14	**ReLu**	-	-
15	**Dropout**	-	0.5
16	**Output**	10	-

overlap. More specifically, class 0 and 1 were assigned to D1, class 2 and 3 to D2, class 4, 5, and 6 to D3, and 7, 8, 9 to D4. Note, if we train a single neural network sequentially: first on the subset of classes 0 and 1, then on classes $2, 3$, next $4, 5, 6$, and finally $7, 8, 9$, the model would suffer catastrophic forgetting. It will forget most of the patterns learned before, and capable of classifying the class corresponding to the last subset (around 28%).

4.2.1. Epochs

One of the most critical hyper-parameters is the amount of local training before aggregation on the centralized server. In this work, we redefine the concept of 'epoch' since it usually refers to the number of times the learning algorithm will work through the whole training dataset. Here we consider mini-batch gradient descent; thus, 'epoch' refers to the number of times the algorithm goes through the mini-batch.

As shown in the Algorithm 4, at the beginning of every round, all the devices have the same model W^t,

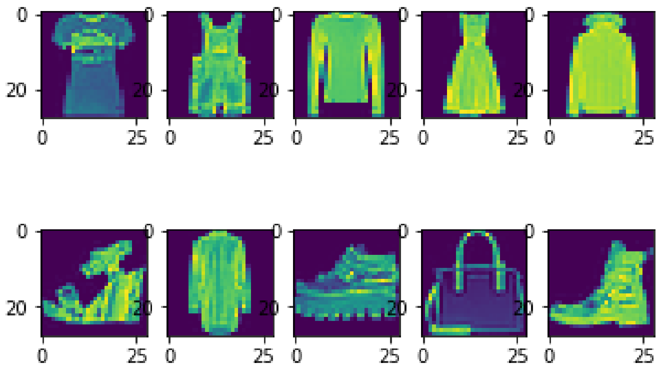
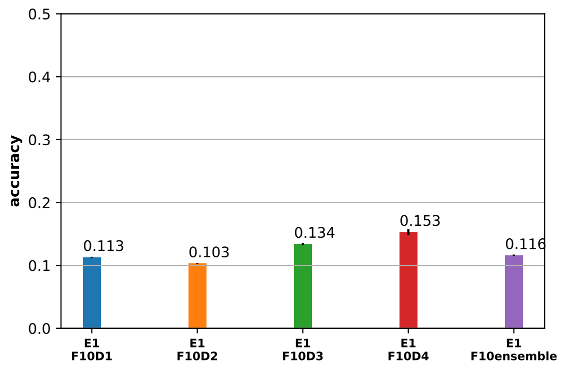


Figure 4: Fashion MNIST dataset: 10 classes, every image has 28 × 28 pixels.

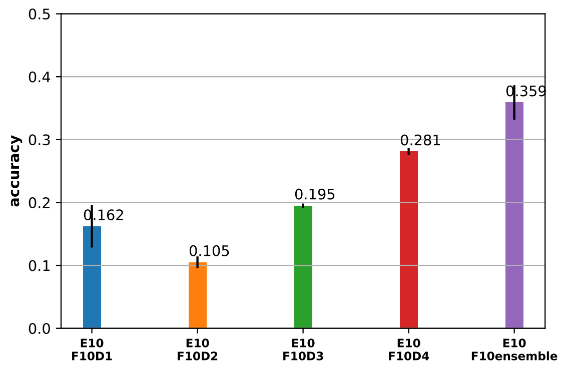
the number of epochs will decide how much variance between updated models $W_1^t, W_2^t, .., W_i^t$, produced by one batch (or multiple batches, determined by aggregation frequency that we will discuss later). If the epoch number is not big enough, the performance after aggregation will not be improved significantly or even be worse. In Figure 5, we plot the accuracy of four distributed models and the aggregated model. Among them, the leftmost four bars represent the accuracy of local models, and the rightmost one is the accuracy of the aggregated model. Note the initial accuracy is 15%. Figure 5a to Figure 5f correspond to different epoch numbers, the accuracy almost monotonically increases with the increment of epoch number, not only for aggregation performance but also for local models. In figure 5f, after enough training, three local models reach the highest accuracy they can, $20\%, 20\%, 30\%$ as they own two, two, and three classes correspondingly.

4.2.2. Aggregation Frequency

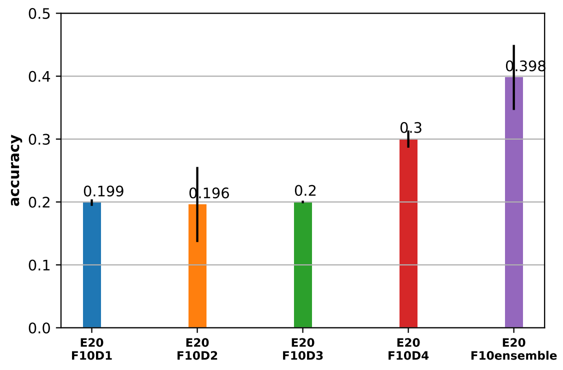
In [10], we only consider one-shot FL (aggregate only once), here we also study the aggregation frequency, which defines the number of acquisitions to train during local training. For instance, assume that we have 10 acquisitions (fixed budget, 400 data for every acquisition), if aggregation frequency is 5, it means that every $10/5 = 2$ acquisitions we aggregate. Note that aggregation frequency is different from epoch: epoch defines the number of repetitions given the acquisition number (training data size), whereas aggregation frequency decides the number of acquisitions, though, both of them are critical factors to enable the performance. If the aggregation frequency is low, we aggregate after a relatively large number of training data, it reduces the communication cost and takes the risk of severe divergence. Instead, if the aggregation frequency is high, we aggregate after a small amount of data, we can avoid the divergence problem, but with increasing the cost of communication. For a given epoch number 45, in Figure 6 we demonstrate the results corresponding to different aggregation frequencies. Correspondingly, we plot the performance concerning different aggregation frequencies (10, 5, 2, and 1). From Figure 6a to Figure 6d, the aggregated accuracy decreases with the decrements of aggregation frequency. We will look at this problem from analyzing the weight divergence



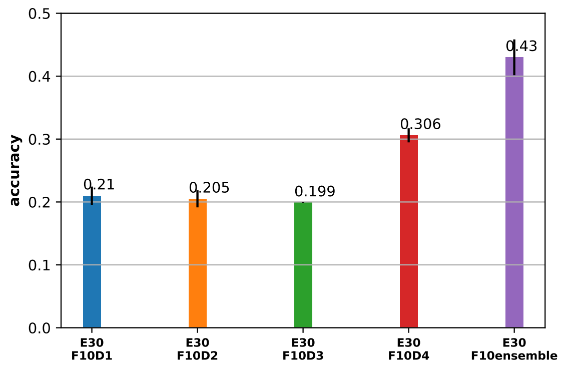
(a) Epoch: 1



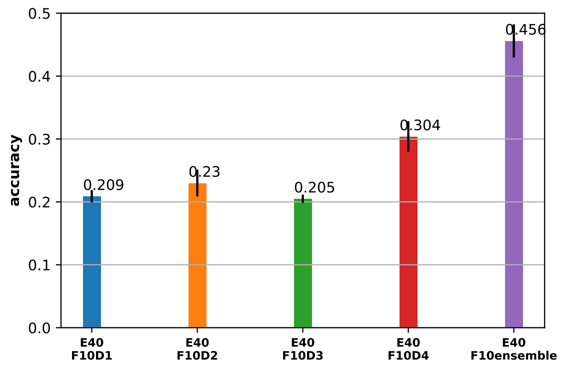
(b) Epoch: 10



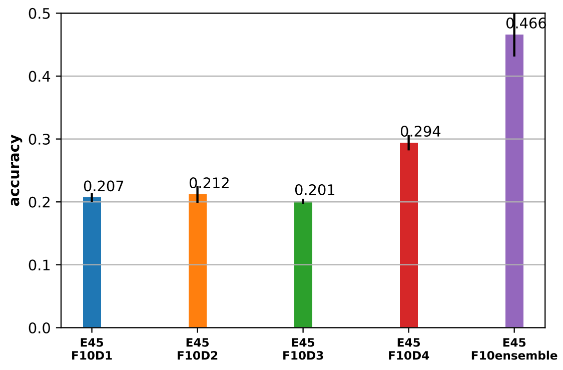
(c) Epoch: 20



(d) Epoch:30

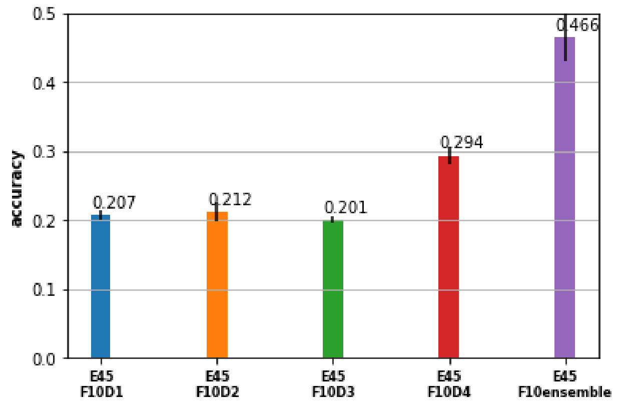


(e) Epoch: 40

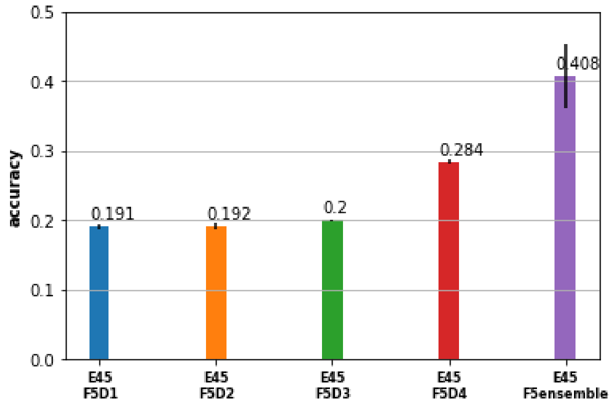


(f) Epoch: 45

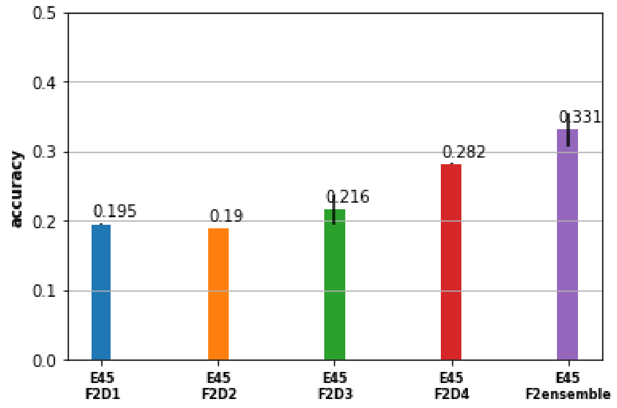
Figure 5: **Epoch Analysis:** The various cases are labeled using the format 'ExFyDz', where 'E' is the epoch, 'F' is the aggregation frequency, and 'D' is the device identifier or the aggregated model respectively. For a given batch, the number of epochs during local training highly influences the aggregation performance. The experimental results show that we should ensure sufficient difference between local models to enable the aggregation effect, which is also related to divergence study in the following experiment.



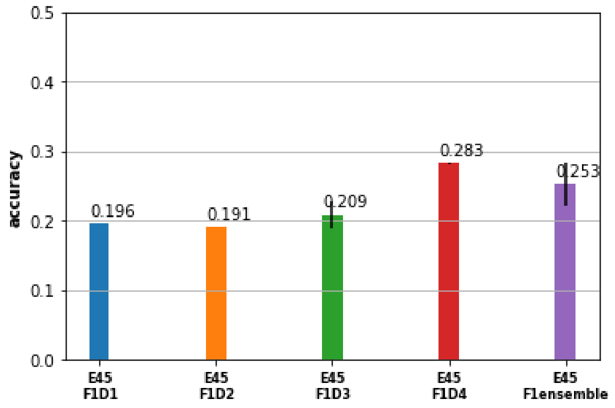
(a) Frequency: 10



(b) Frequency: 5



(c) Frequency: 2



(d) Frequency: 1

Figure 6: **Aggregation Frequency Analysis:** The various cases are labeled using the format 'ExFyDz', where 'E' is the epoch, 'F' is the aggregation frequency, and 'D' is the device or the aggregated model respectively. For a given Epoch 45 and 10 acquisitions of data, we plot the performance with respect to different aggregation frequenciesHigh aggregation frequency has higher accuracy, increasing the cost of communication, and vice versa.

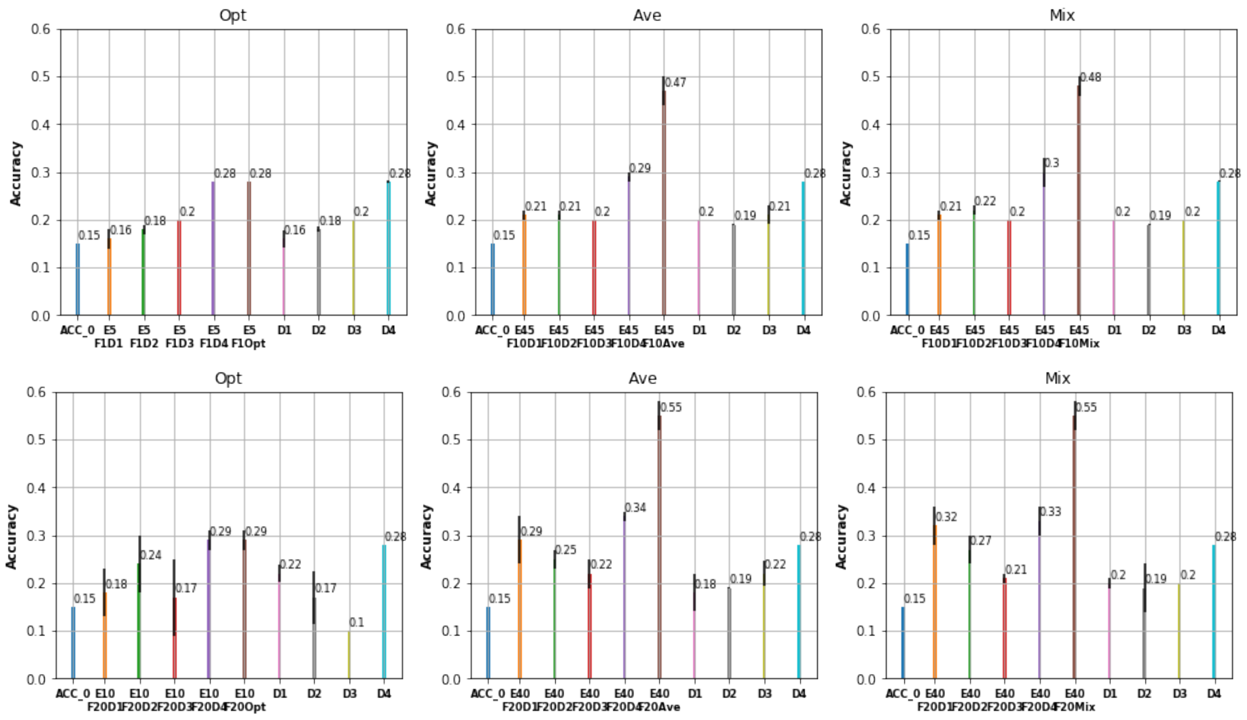


Figure 7: **Aggregation strategies:** We compare accuracy with different aggregation strategies, namely *AveFL*, *OptFL* and *MixFL* (top: 10 acquisitions, bottom: 20 acquisitions). The various cases are labeled using the format 'ExFyDz', where 'E' is the epoch, 'F' is the aggregation frequency, and 'D' is the device or the aggregation model respectively. ACC_0 is the initial accuracy. The rightmost four bars are the performance by the independent local models without considering aggregation.

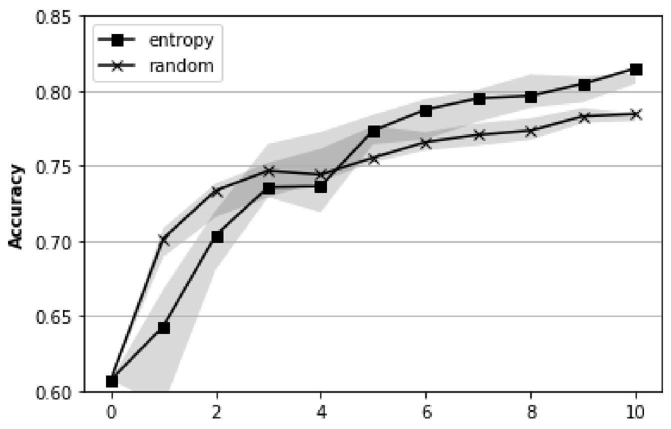


Figure 8: Given the same number of data, AL outperforms random choice in terms of accuracy.

in Section 4.5. Both the epoch and the aggregation frequency cause divergence, but aggregation has more significant impact than the epoch number.

4.3. Aggregation Strategies

205 We consider three aggregation strategies in this paper: *AveFL*, *OptFL* and *MixFL*. As we discussed in the previous section, *AveFL* averages the parameters of models during aggregation; *OptFL* opts for the model with optimal performance; and *MixFL* is the mixture of *AveFL* and *OptFL*, in each iteration it chooses the better one between them. The result (shown in Figure 7) demonstrates that there is no big difference between *OptFL* and *MixFL* for both cases of 10 and 20 acquisitions. However, the whole distribution they learned is
210 different, as we discuss in Section 4.6.

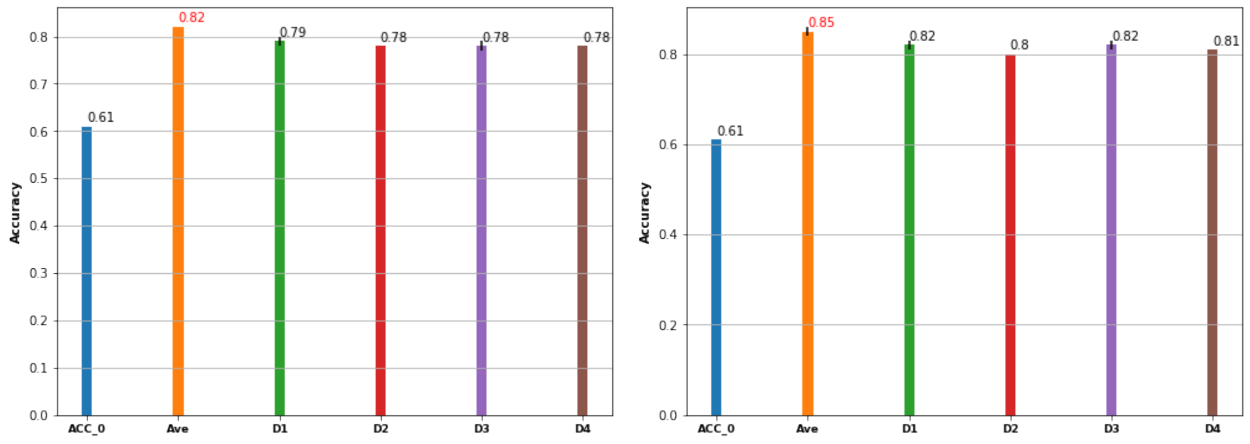
4.4. Non-IID Data Type ii *(FL with AL)*

For Non-IID Type ii, we simulate it by applying AL on the four edge devices. AL can be considered an effective way of choosing data than random sampling, and this behavior causes a slightly biased data generation. For instance, in [10], we select the data with maximal entropy (uncertainty) to train our model.
215 The method is shown in Algorithm 5. In Figure 8, we firstly show that AL outperforms random choice in terms of prediction accuracy. Also, in Figure 9 shows how aggregation affects the performance when FL combines AL.

4.5. Weights Divergence and Aggregation Performance

In this subsection, we quantitatively investigate the correlation between weight divergence and aggregation performance. Firstly, let us define the divergence of layer l of device i as follows:

$$\text{divergence}_i^l = \frac{|W_i^{tl} - W_{\text{aggregated}}^{tl}|}{|W_i^{tl}|}$$



(a) Random

(b) Entropy

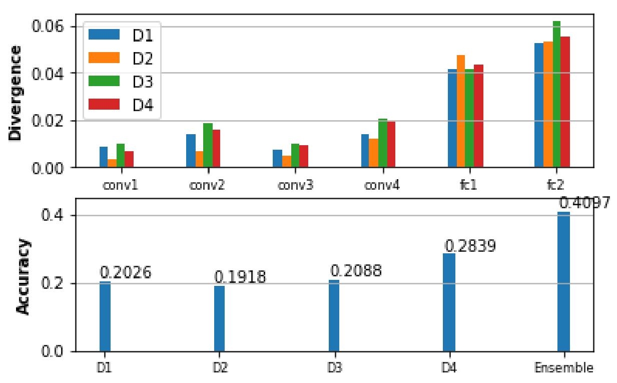
Figure 9: From left to right we plot initial accuracy, aggregation accuracy and four model accuracy without aggregation step. First of all, no matter random or AL, the result of aggregated model has higher accuracy compared with no aggregation. Overall, AL has better performance with respect to random choice (from the second bar to the last bar).

Where W_i^{tl} is the layer l of model (device) i at iteration t and $W_{\text{aggregated}}^{tl}$ is the the layer l of aggregated model at iteration t.

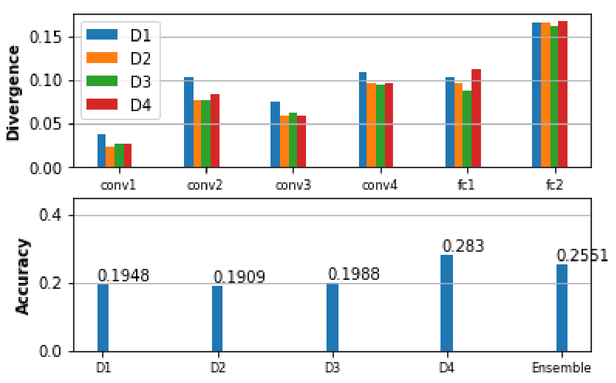
In Figure 10, we plot the divergence of all layers in the network (above) and the corresponding aggregated result (below). The first coordinate represents the network layers, and the four colored bars represent the different devices. From Figure 10a to Figure 10d, the divergence decreases for all the layers; however, the aggregation increases in the beginning and stops increasing at some point. It could indicate that if the divergence value is too large (Figure 10a), the aggregation effect is not fully enabled, and on the other hand, if it is too small (Figure 10d), it may disable the aggregation effect. Our result is consistent with [9], where they also showed the accuracy reduction is significantly correlated with weight divergence.

4.6. Correlation between Local Models and Aggregated Model

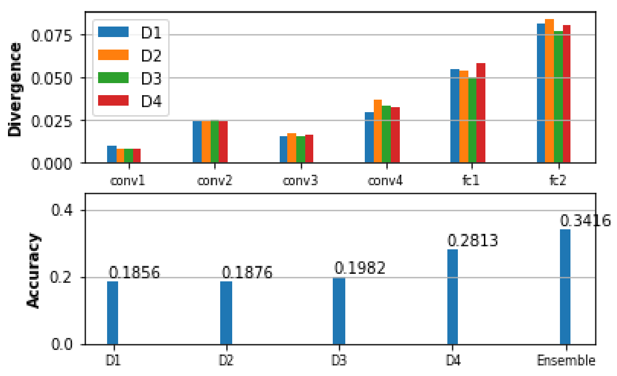
We can also consider the aggregated model as the Gaussian Mixture Model (GMM) [26]. Suppose we have C classes, which correspond to C models $M_1, M_2, .., M_C$. We define GMM as $M_{GMM} = \sum_{i=1}^{C} \alpha_i M_i$ and $\sum_{i=1}^{C} \alpha_i = 1$. In our case, we consider α uniformly distributed since we do not have prior knowledge and do not want to solve the optimization problem to compute α_i. It implies an assumption that the ten classes share some common features. Averaging weights is like partially 'copying' the classification capability of different classes from their related edge devices. We call it 'partially' because models from other categories will dilute the effect. The experimental evidence is shown in Figure 11. For *AveFL* (Figure 11a), we plot the histogram of correctly classified classes for four edge devices (corresponds to four colors) in the left figure and the histogram of correctly classified classes for the average model in the right figure. As we can see, without



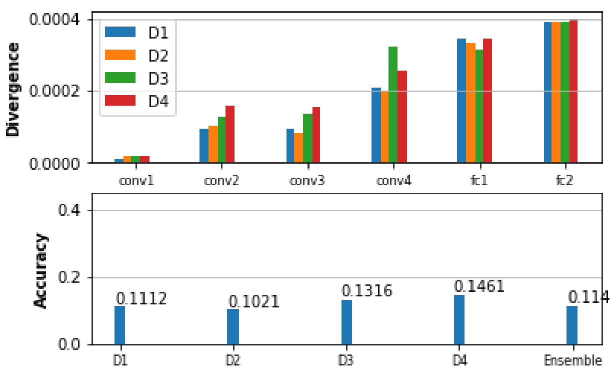
(a) Divergence Level i



(b) Divergence Level ii



(c) Divergence Level iii

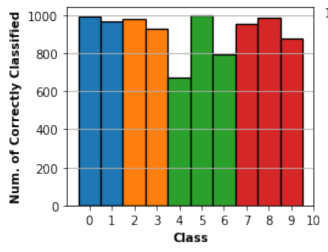


(d) Divergence Level iv

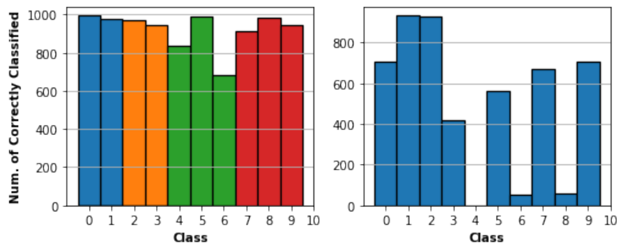
Figure 10: Plot above represent the divergence and plot below is the corresponding accuracy of local models and aggregated model. X coordinator of plot (above) indicates the layers of model and bars with different colors represent the different devices. Aggregation Effect has high correlation with divergence grade. Here we compared four level of divergence, From Fig 10a to Figure 10d the divergence decreases, Figure 10c corresponds to best aggregation performance. **Note that y coordinators are not aligned due to the different magnitudes.**

aggregation the local model can **only** predict their corresponding categories. For instance, d_1 generates class 0 and 1, the trained model on d_1 can only predict 0 and 1. However, the prediction by aggregated model can cover most of the classes, except with difficulty in classifying 4 and 6. In Figure 4, we can see class 4 is 'coat' and class 6 corresponds to 'shirt'(label starts from 0). These two classes are very similar, and it is not easy to distinguish. While, *MixFL* (Figure 11b) has different behavior: it learns different distributions from aggregation, though, their overall accuracy is similar (shown in Figure 7).

To further study how local models benefit the aggregated model, we analyze the neuron activation patterns. We choose 10 test images from class 1 and feed them into local model trained by d_1 in Figure 12a, aggregated model in Figure 12b and local model from d_3 in Figure 12c. The x coordinator indicates that the nodes of the fully connected layer right before the softmax layer, and the heatmap values are the output of the nodes.

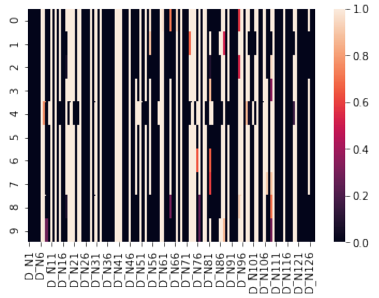


(a) *AveFL*

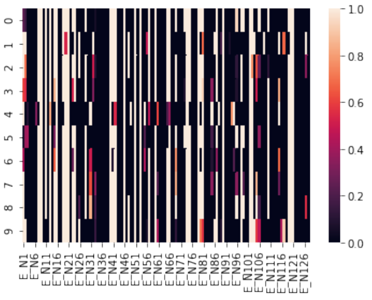


(b) *MixFL*

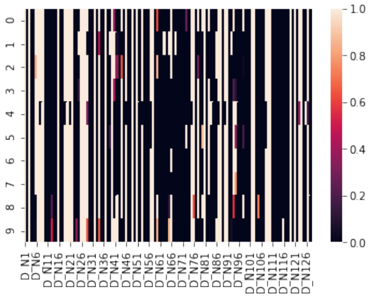
Figure 11: Figure 11a is the correctly classified histogram of Average aggregation case, in the left plot the four different color bars correspond to four models of edge devices without aggregation. They can only correctly classify their own categories. While the right plot in Figure 11a has a better comprehensive capability of classification, it has the difficulty of distinguishing classes 4 and 6. In Figure 11b we plot the same results for the mix aggregated method.



(a) Activation of Model from d_1



(b) Activation of Ensem. Model



(c) Activation of Model from d_3

Figure 12: We choose 10 test images from class one and feed into local model from d_1 in Figure 12a, aggregated model in Figure 12b and local model from d_3 in 12c. The x coordinator indicates the nodes of the fully connected layer right before the softmax layer, and the heatmap values are the output of the nodes. The activation of pattern of d_1 is similar to the aggregated model, fairly different from d_3 where class 1 is not generated.

The activation of the pattern of d_1 is similar to the aggregated model, fairly different from d_3 that does not have any information about class 1.

5. Conclusion and Future Work

Distributed machine learning has several virtues, including the potential to reduce data aggregation and thus improved privacy. However, this virtue poses a potential challenge, namely that the edge devices are set to learn from Non-IID data. Hence, to investigate the robustness of Federated Learning to Non-IID data, we simulate two scenarios. Furthermore we analyze and compare different aggregation strategies: *AveFL*, *OptFL* and *MixFL*. We presented evidence that federated learning is robust to sampling bias, and also we found that the epoch (amount of local learning) and the aggregation frequency are important parameters for Federated Learning. In the end, we also post-process the prediction performance to understand the correlation between local models and the aggregated model.

Acknowledgments

The research leading to these results has received funding from the European Union's Horizon 2020 research and innovation program under the Marie Sklodowska-Curie grant agreement No. 764785, FORA-Fog Computing for Robotics and Industrial Automation.

References

[1] J. Konečnỳ, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, D. Bacon, Federated learning: Strategies for improving communication efficiency, arXiv preprint arXiv:1610.05492.

[2] M. Barzegaran, N. Desai, J. Qian, K. Tange, B. Zarrin, P. Pop, J. Kuusela, Fogification of electric drives: An industrial use case, in: The 25th International Conference on Emerging Technologies and Factory Automation ETFA2020, 08 Sep 2020, Vienna, Austria, 2020.

[3] J. Qian, P. Tiwari, S. P. Gochhayat, H. M. Pandey, A noble double dictionary based ecg compression technique for ioth, IEEE Internet of Things Journal.

[4] C. He, C. Tan, H. Tang, S. Qiu, J. Liu, Central server free federated learning over single-sided trust social networks, arXiv preprint arXiv:1910.04956.

[5] Y. Zhao, C. Yu, P. Zhao, H. Tang, S. Qiu, J. Liu, Decentralized online learning: Take benefits from others' data without sharing your own to track global trend, arXiv preprint arXiv:1901.10593.

[6] A. A. Diro, N. Chilamkurti, Distributed attack detection scheme using deep learning approach for internet of things, Future Generation Computer Systems 82 (2018) 761–768.

[7] B. Lakshminarayanan, A. Pritzel, C. Blundell, Simple and scalable predictive uncertainty estimation using deep ensembles, in: Advances in neural information processing systems, 2017, pp. 6402–6413.

[8] H. Zhu, Y. Jin, Multi-objective evolutionary federated learning, IEEE transactions on neural networks and learning systems.

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

[10] J. Qian, S. Sengupta, L. K. Hansen, Active learning solution on distributed edge computing, arXiv preprint arXiv:1906.10718.

[11] J. Qian, S. P. Gochhayat, L. K. Hansen, Distributed active learning strategies on edge computing, in: 2019 6th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/2019 5th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom), IEEE, 2019, pp. 221–226.

[12] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, Dropout: a simple way to prevent neural networks from overfitting, The journal of machine learning research 15 (1) (2014) 1929–1958.

[13] L. Breiman, Bias, variance, and arcing classifiers, Tech. rep., Tech. Rep. 460, Statistics Department, University of California, Berkeley … (1996).

[14] Z.-H. Zhou, Ensemble methods: foundations and algorithms, CRC press, 2012.

[15] T. Chen, T. He, M. Benesty, V. Khotilovich, Y. Tang, Xgboost: extreme gradient boosting, R package version 0.4-2 (2015) 1–4.

[16] Z. Xu, G. Huang, K. Q. Weinberger, A. X. Zheng, Gradient boosted feature selection, in: Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining, 2014, pp. 522–531.

[17] D. A. Klein, A. B. Cremers, Boosting scalable gradient features for adaptive real-time tracking, in: 2011 IEEE International Conference on Robotics and Automation, IEEE, 2011, pp. 4411–4416.

[18] J. Son, I. Jung, K. Park, B. Han, Tracking-by-segmentation with online gradient boosting decision tree, in: Proceedings of the IEEE International Conference on Computer Vision, 2015, pp. 3056–3064.

[19] T. Nishio, R. Yonetani, Client selection for federated learning with heterogeneous resources in mobile edge, in: ICC 2019-2019 IEEE International Conference on Communications (ICC), IEEE, 2019, pp. 1–7.

[20] P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, K. Bonawitz, Z. Charles, G. Cormode, R. Cummings, et al., Advances and open problems in federated learning, arXiv preprint arXiv:1912.04977.

[21] T. B. Hashimoto, M. Srivastava, H. Namkoong, P. Liang, Fairness without demographics in repeated loss minimization, arXiv preprint arXiv:1806.08010.

[22] A. Krogh, J. A. Hertz, A simple weight decay can improve generalization, in: Advances in neural information processing systems, 1992, pp. 950–957.

[23] R. Pinsler, J. Gordon, E. Nalisnick, J. M. Hernández-Lobato, Bayesian batch active learning as sparse subset approximation, in: Advances in Neural Information Processing Systems, 2019, pp. 6356–6367.

[24] S. S. Sarwar, A. Ankit, K. Roy, Incremental learning in deep convolutional neural networks using partial network sharing, IEEE Access.

[25] S. Ioffe, C. Szegedy, Batch normalization: Accelerating deep network training by reducing internal covariate shift, arXiv preprint arXiv:1502.03167.

[26] J. J. Verbeek, N. Vlassis, B. Kröse, Efficient greedy learning of gaussian mixture models, Neural computation 15 (2) (2003) 469–485.