Towards Federated Learning: Robustness to Extremely Unbalanced Data for Edge Computing

Jia Qian, Xenofon Fafoutis, *Member, IEEE*, Lars Kai Hansen

***Abstract*—Federated Learning allows remote centralized server training models without to access the data stored in distributed (edge) devices. Most work assume the data generated from edge devices is identically and independently sampled from a common population distribution. However, such ideal sampling may not be realistic in many contexts where edge devices correspond to units in variable context. 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, we investigate two such scenarios. First, we study Federated Learning of a classifier from data with edge device class distribution heterogeneity. Second, we study Federate Learning of a classifier with active sampling at the edge. We present evidence in both scenarios, that federated learning is robust to class imbalance.**

***Index Terms*—Edge Computing, Fog Computing, Active Learning, Federated Learning, Distributed Machine Learning, User Data Privacy**

I. Introduction

Federated Learning (FL) [1] 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 realised). From a practical point of view an Federated Learning offers unique opportunities for data protection. In particular, Federated Learning can be realised without aggregating the training data but rather the data remains in its generation location, which provides the opportunity to secure user privacy. Extending FL to other machine learning paradigms, including reinforcement learning, semi-supervised and unsupervised learning, active learning, and online learning [8], [9] all present interesting and open challenges. Most of the research work assume that data is **Independent** and **Identically Distributed** (IID) on the distributed edge devices. This 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, hence biased. A high level depiction of this scenario is presented in Figure 1.

J. Qian, X. Fafoutis and L. K. Hansen are with Department of Applied Mathematics and Computer Science, Technical University of Denmark, 2800 Lyngby, Denmark (e-mail: jiaq, xefa, lkai}@dtu.dk). The research leading to these results has received funding from the European Unions Horizon 2020 research and innovation program under the Marie Sklodowska-Curie grant agreement No. 764785, FORA-Fog Computing for Robotics and Industrial Automation. This work was further supported by the Danish Innovation Foundation through the DanishCenter for Big Data Analytics and Innovation (DABAI).

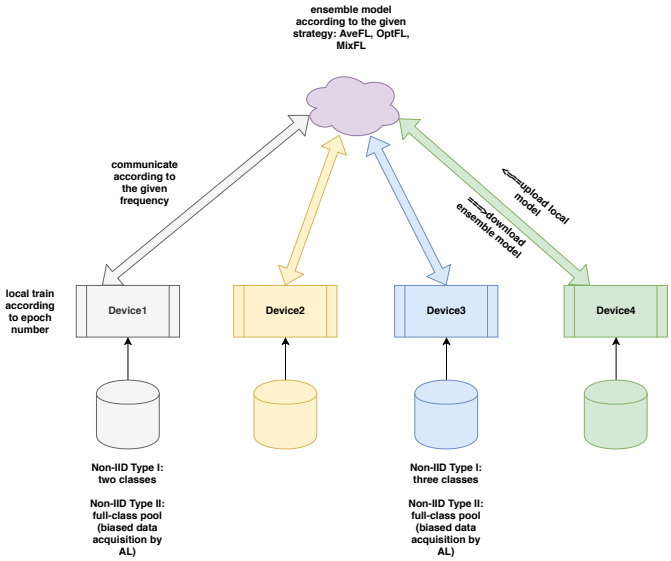


Fig. 1: Federated Learning Scheme.

As a first investigation into the robustness of distributed machine learning, 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; Type II we equip the edge devices with active learning capacity, hence the ability actively sample useful data from a local pool of candidates.

Our contribution can be summarized as:

- We simulate two types of Non-IID data generation and study how do training time and communication level influence the learning.
- Investigate the relationship between edge device diversity and federated learning performance.
- Investigate the effects of federation by analysis of deep network activation patterns.

The remainder of this paper is organized as follows. Section II explains preliminary concepts and the related work, In Section III, we introduce the proposed scheme. Section IV covers the details of our experimental design and data collection strategy followed by a discussion on our results. Section V concludes the paper.

II. Preliminaries and Related Work

A. Federated Learning

Federated learning (FL) is a collaborative form of machine learning where the training process is distributed among many

users enabling machine learning without complete centralized access to training data [1]. In FL, the data remains at its generated location, which has the potential of privacy and reduces transmission costs. In principle, this idea can be applied to any model for which the criterion of updates can be defined, which naturally includes the methods based on stochastic gradient descent, which is used to train, for instance, linear regression, logistic regression, neural networks, and linear support vector machines [2] [3].

FL was first proposed by [1] for the user privacy consideration in mobile networks, and it is a very useful framework in edge computing. Since its original publications, a series of works employ FL: [5] detects attacks using a distributed network, [6] predicts model uncertainty by a deep ensemble model, and [7] aims to optimize the structure of Neural network in FL. Most of the FL applications assumes the data is IID on edge devices. Only [4] 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 the related work, 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 any repetition between devices, (ii) Type II whereby we employ AL on the edge devices to simulate a slightly biased data generation.

B. Active learning of neural networks

Labeling may be challenging and expensive. Considering edge devices related to diverse set of users the labeled data sets can be skewed towards certain classes and thus without exchange with other users user models can be highly biased. It is therefore natural to combine between Federated Learning (FL) and Active Learning (AL) as Non-IID Type II, and we reported the prototype (tested on MNIST dataset [20]) in [11]. Typically, AL achieves higher performance with the same number of data samples or achieves a given performance using fewer data. It can be generally divided into two categories: pool-based and stream-based. The stream-based Active Learning 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 labeled dataset. More specifically, the initially trained model carefully selects instances from the unlabeled pool according to the acquisition function and then asks the oracle to label them. Sequentially, we include the labeled ones to the training set for further training. We can repeat such operations for several times. In our previous work [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 III-C). To apply AL on a Neural Network, we build a Stochastic Neural Network through a free ride Dropout [25]. The Stochastic Neural Network can be considered as a model, which outputs different

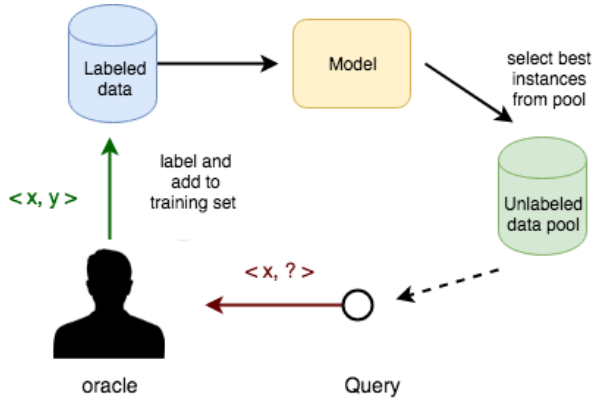


Fig. 2: Pool-based Active Learning Scheme.

values for the same input fed in the model several times. More details can be found in [11].

C. The Boosting approach

Boosting is an ensemble meta-algorithm for primarily reducing bias, and also variance [23] in supervised learning, and a family of machine learning algorithms that convert weak learners to strong ones [24]. The approach of the present work can be related to boosting by viewing the ensemble as a combination of 'weak' (specialized) edge models in repeated steps of federation. In [12] the authors proposed the Boosting Gradient Classifier, which has a set of week 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 in function. Analogously, we can also make it dynamic like boosting gradient classifier. 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 [13]–[15]. 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.

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

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

Algorithm 1 *AveFL*

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

Algorithm 2 *OptFL*

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

Algorithm 3 *MixFL*

Input: local models at round t W_j^t
Output: ensemble model W^t
$W_{\text{ave}}^t = \textit{AveFL}(W_j^t)$
$W_{\text{opt}}^t = \textit{OptFL}(W_j^t)$
$\text{acc}_{\text{ave}} = A(\text{BNN}(X, W_{\text{ave}}^t))$
$\text{acc}_{\text{opt}} = A(\text{BNN}(X, W_{\text{opt}}^t))$
if $\text{acc}_{ave} >= \text{acc}_{opt}$ **then**
    Return W_{ave}^t
else
    Return W_{opt}^t

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, the Boosting Gradient classifier and federation attempt to improve the performance by assembling a set of weak models. The Boosting Gradient classifier works on a dataset with extracted features, specifically, decide 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.

III. Proposed scheme

A. Federated Learning

We define the goal of FL as learning a model with parameters embodied in matrix from data stored across a large number of clients (Edge Devices). Suppose the server distributes the model (at round t) W_t to n devices for further updating, and the updated models are denoted as $W_t^1, W_t^2, W_t^3, ...W_t^n$. Then, the devices send the updated models back to the server, and the server updates the model W according to the aggregated information:

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

where combination weights α 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 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, labelled *OptFL* (Algorithm 2). In Section IV, 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 ensemble 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 ensemble is defined in Equation 4.

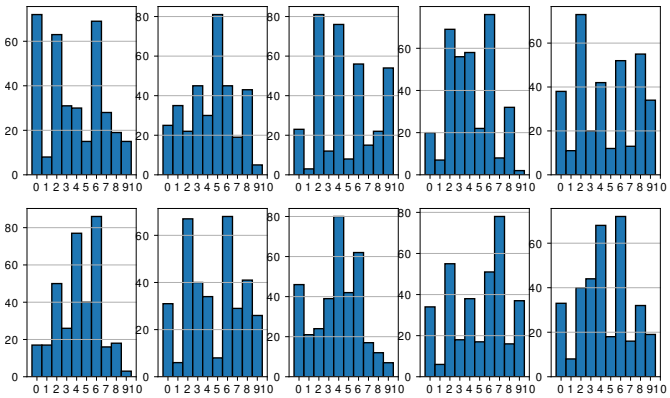


Fig. 3: Biased Data Acquisition by Active Learning: we demonstrate the distribution of data acquired by AL for 10 acquisitions. They are unbalanced in different ways from acquisition to acquisition, even when the data in pools are class balanced.

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

Ensemble Weights:

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

Algorithm 4 Non-IID Type I

1: **if** t==0 **then**
2: set initial training images number $m = 400$
3: form initial training set X_0 with size equal to m
4: train model W_0 by X_0
5: dispatch model W_0 to device $D_1, D_2, D_3, ..., D_n$
6: **else**
7: **for** j=1,2,..,n **do** n devices (in Parallel)
8: X^t_{train} = RandomSample()
9: W^t_j = Train(X^t_{train})
10: **end**
11: W^{t+1} = *AveFL*(W^t_j) for j=1,2,...,n
12: $\mathrm{W}^{t+1}_j = W^{t+1}$
13: **end**
14: **end**

Ensemble 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 damaging weight divergence can occur. If initial models are different, they might be placed in a different low cost regions of the cost landscape. Thus, after the ensemble averaging step it might be sub-optimal. In this paper, we also aim to investigate how does the amount of local training influence the result, we decide to work on the weights ensemble for the sake of flexibility.

B. Method for Non-IID Type I

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

1) **Initialization:** At initialization, the centralized server trains an initial model W^0 using m data samples. We denote the model as W^t, where t is the number of current round.
2) **Distribution:** Server dispatches the model W^t to n activated edge devices $D_1, D_2, ..., D_n$.
3) **Local Training:** All edge devices implement local training and update their models $W^t_1, W^t_2, ..., W^t_n$. This step incorporates one or multiple cycles of data acquisition.
4) **Ensemble:** Edge devices transmit their corresponding models to server and the server ensembles $W^t_i, i = 1, 2.., n$ to get W^{t+1}. The ensemble could be *AveFL*, *OptFL* or *MixFL*.
5) Repeat steps 2-4 if necessary.

The specific algorithm is described in Algorithm 4.

Algorithm 5 Non-IID Type II

1: **if** t==0 **then**
2: set initial training samples number $m = 1000$
3: form initial training set X_0 with size equal to m
4: train model W_0 by X_0
5: dispatch model W_0 to device $D_1, D_2, D_3, ..., D_n$
6: store 50 samples, label as X_{50}
7: **else**
8: **for** j=1,2,..,n **do** n devices (in Parallel)
9: **for** t=1,2..T **do**
10: **for** $i = 1$ to $poolsize$ **do**
11: $\log p(x_i)$, $p(x_i)$ = BNN^{t-1}_j (x_i)
12: **end**
13: $[x^t_1, x^t_2, ..x^t_{350}]$ = Acquisition($\log p(x_i), p(x_i)$)
14: $X^t_{train} = X_{50} \cup [x^t_1, x^t_2, ..x^t_{350}]$
15: BNN^t_j = Train(X^t_{train})
16: **end**
17: M^{t+1} = Ensemble(BNN^t_j) for j=1,2,...,n
18: $\mathrm{BNN}^{t+1}_j = M^{t+1}$
19: **end**
20: **end**

C. 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 can lead to biased samples. First, we divide whole training set into 4 parts, one part for one edge device. Then we build a pool with 4000 images randomly sampled from one part for the computation consideration since we need to measure the uncertainty of every data in the pool. We can consider the pool as a balanced set, but when we acquire data from pool to label is unbalanced, one instance shown in Figure 3. For scalability, in this paper, we perform an online AL (Algorithm 5, line 14). Namely, the model is further trained by the new batch, without the access of the old data (except 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 forgetting the problem of online learning by a cheap trick, storing 50 images, which is a balanced set (5 images per class) and will be combined with a new batch to train the model. After complete current-round training, we dump the new batch and only keep 50 images in labeled set in Figure 2. Moreover, we also use weight decay [22] as a regularize 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.

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

[17] learns the weights that can mostly approximate the distribution of the data from pool (shown in Figure 2) by solving an optimization problem. It is highly computation-

demanding and not suitable for edge devices. There are some work [18] that attempt to avoid forgetting by dividing the Neural Network architecture into parts and assign them to different edge devices, but it requires restrict synchronization during ensemble. Our Non-IID Type II method is sketched as:

1) **Initialization:** In the beginning, centralized server trains an initial model W^0 using m data samples. To generalize, we denote the model by W^t, where t is the number of current round.
2) **Distribution:** The central server dispatches 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 Stochastic Neural Network approximated by Dropout [25], 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) **Ensemble:** Edge devices transmit their corresponding models to server and the server ensembles $W_i^t, i = 1, 2.., n$ to get W^{t+1}. The ensemble 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.

D. Architecture

The deep learning model is composed of four convolutional layers and one fully-connected layer and a softmax layer shown in Table I. Note that we did not use batch normalization

TABLE I: 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	-

[16] in the architecture. Mathematically, it defines as shown in Equation 6 and Equation 7. Suppose we have a batch $B = x_1, x_2, ..., x_m$, then it is normalized (equation 4) and next we infer new mean (β) and variance (γ) in training process. By doing so, we can limit the variance between batches. In the Non-IID case, the means and variances optimized in the local training stage corresponding to their actual samples, hence not suitable during the ensemble stage. This is very critical to enable ensemble effect in highly biased data generation,

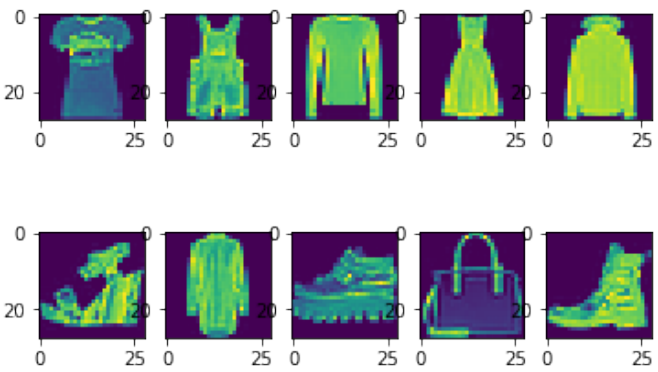


Fig. 4: Fashion MNIST dataset: 10 classes, every image has 28×28 pixels.

otherwise, the ensemble model performs very poorly (e.g. 20% accuracy whereas without batch normalization 47%).

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

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

IV. Experiments and Results

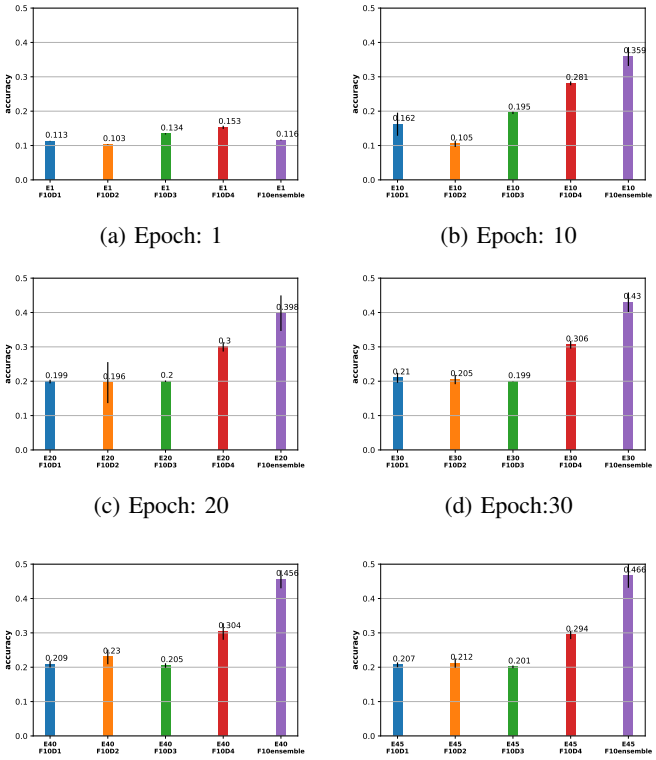
In this section, we discuss the experimental setup along with the dataset used for our evaluation, and the results of these experiments.

A. Dataset

Fashion-MNIST (shown in Figure 4) is published by Zalando, as the alternative of MNIST ([20]) dataset. It consists of a training set of 60,000 examples and a test set of 10,000 examples. Each example has 28×28 pixels, associated with a label for 10 classes. Each pixel has a single pixel-value associated with it, indicating the lightness or darkness of that pixel, with higher numbers meaning darker. This pixel-value is an integer between 0 and 255.

B. 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 overlap. In particular class $0, 1$ was assigned to D1, class $2, 3$ to D2, class $4, 5, 6$ to D3 and $7, 8, 9$ to D4. Note, if we trained a single deep network sequentially: First on the subset of classes $0, 1$, then on classes $2, 3$, next $4, 5, 6$, and finally on $7, 8, 9$, 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%).



(a) Epoch: 1 (b) Epoch: 10

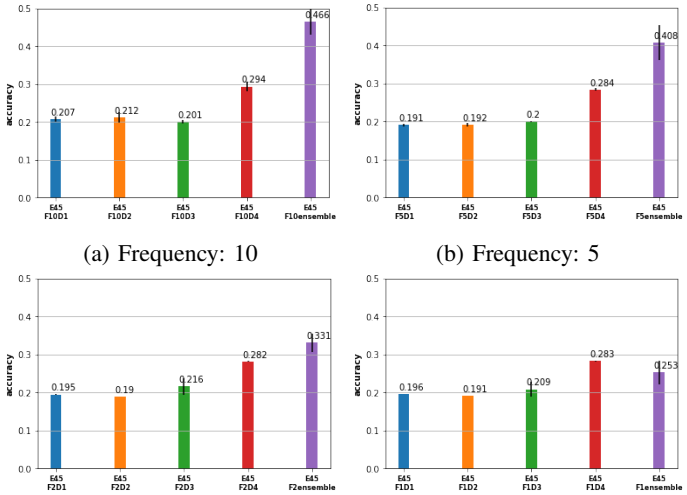
(c) Epoch: 20 (d) Epoch:30

(e) Epoch: 40 (f) Epoch: 45

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

1) Epochs: One of the most important hyper-parameters is the amount of local training before ensembling on the centralized server. In this work, we redefine the concept of 'epoch' since normally it refers to the number times that the learning algorithm will work through the entire training dataset. Since we consider mini-batch gradient descent, in this paper, 'epoch' refers to the number times that the algorithm goes through the mini-batch before the ensemble aggregation.

As shown in the Algorithm 4, at the beginning of every round, all the devices have the identical model W^t, the number of epochs will decide how much variance between updated models $W_1^t, W_2^t, .., W_i^t$, which is produced by one batch (or multiple batches, decide by ensemble frequency that we will discuss later). If the epoch number is not big enough, after ensemble there will not have a big difference. In Figure 5, we plot the accuracy of four distributed models and the ensemble model. Among them, the leftmost four bars represent the



(a) Frequency: 10 (b) Frequency: 5

(c) Frequency: 2 (d) Frequency: 1

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

accuracy of local models and the rightmost one is the accuracy of the ensemble model. Note the initial accuracy is 15%. Figure 5a to Figure 5f correspond to different epoch numbers, the accuracy almost monotonically increase with the increment of epoch number, no only for ensemble 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.

2) Ensemble Frequency: In [11] we only consider one-shot FL (ensemble only once), here we also study the ensemble frequency, which defines the number of acquisitions to train before ensemble. For instance, assume that we totally have 10 acquisitions (fixed budget, 400 data for every acquisition), if ensemble frequency is 5, it means that every $10/5 = 2$ acquisitions we ensemble. Note that ensemble frequency is different from epoch: epoch defines the number of repetition given the acquisition number (training data size), whereas ensemble frequency decides the number of acquisitions, though, both of them are critical factors to enable ensemble effect. If the ensemble frequency is low, we ensemble after a quite large number of training data, it reduces the communication cost but also takes the risk of severe divergence. Instead, if the ensemble frequency is high, we ensemble after a fewer batches, thus, we can avoid the divergence problem increasing the cost of communication. For a given epoch number 45, in Figure 6 we demonstrate the results corresponding to different ensemble frequencies. Correspondingly, we plot the performance with respect to different ensemble frequencies (10, 5,

2 and 1). From Figure 6a to Figure 6d, the ensemble accuracy decreases with the decrements of ensemble frequency. We will look at this problem from analyzing the weight divergence in Section IV-E. Both the epoch and the ensemble frequency cause divergence, but ensemble has more significant effect than the epoch number.

C. Ensemble Strategies

We consider three ensemble strategies in this paper: *AveFL*, *OptFL* and *MixFL*. As we discussed in the previous section, *AveFL* averages the parameters of models during ensemble; *OptFL* opts for the model with optimal performance; and *MixFL* is the mixture of *AveFL* and *OptFL*, in each iteration choose the better one between them. The result, shown in Figure 7, demonstrate 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 different, as we discuss in Section IV-F.

D. 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 as an effective way of choosing data than random sampling, and this behavior causes a slightly biased data generation. For instance, in [11] we choose the data with maximal entropy (uncertainty) to train our model. The method is shown in Algorithm 5. In Figure 8, we firstly show that AL outperforms random choice in terms of prediction accuracy. In addition, in Figure 9 shows how does the ensemble affects the performance when FL combines AL.

E. Weights Divergence and Ensemble Performance

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

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

Where W_i^{tl} is the layer l of model (device) i at iteration t and W_{ensemble}^{tl} is the the layer l of ensemble model at iteration t. In Figure 10 we plot the divergence of all layers in the network (above) and the corresponding ensemble result (below). The first coordinate are 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 ensemble increases in the beginning and stop increasing at some point. It could indicate that if the divergence value is too large (Figure 10a), the ensemble effect is not fully enabled; and on the other hand, if too small (Figure 10d), it may disable the ensemble effect. Our result are consistent with [4], where they also showed the accuracy reduction can be explained by weight divergence.

F. Correlation between the device models and ensemble model

We can also consider the ensemble model as the Gaussian Mixture Model (GMM) [19]. 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, otherwise, it does not make sense. Averaging weights is like partially 'copying' the classification capability of different classes from their related edge devices. We call it 'partially' because the effect will be diluted by models from other categories. 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 ensemble the local model can **only** predict the categories generated from their own. For instance, D_1 generates class 0 and 1, the trained model on D1 can only predict 0 and 1. However, the ensemble model can predict most of the classes, except with the difficulty to classify 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 difficult to distinguish, particularly when the minor different between two classes has been neutralized after ensemble. While, *MixFL* (Figure 11b) has different behavior: it learns different distributions from ensemble, though, their overall accuracy is similar (shown in Figure 7).

To further study how local models benefit the ensemble model, we analyze the neuron activation patterns. We choose 10 test images from class 1 and feed them into local model from D1 in Figure 12a, ensemble model in Figure 12b and local model from D3 in Figure 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 D1 is similar to the ensemble model, fairly different from D3 that does not have any clue about class 1.

V. Conclusion

Distributed machine learning has a number of virtues including the potential to reduce data aggregation and thus improved privacy. This virtue, however, 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 ensemble 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 ensemble 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 ensemble model.

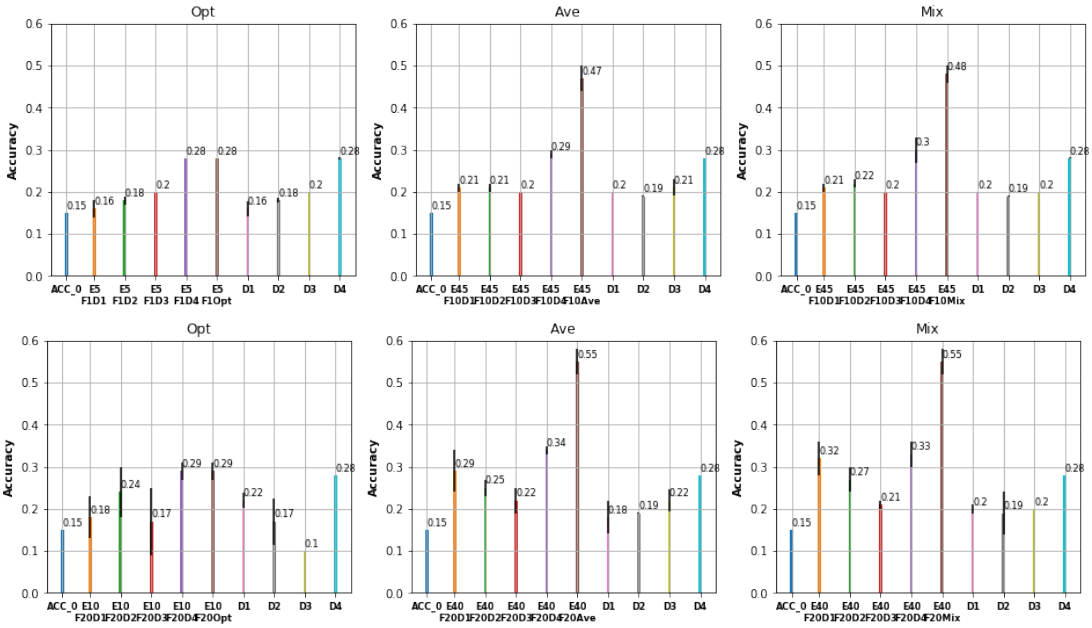


Fig. 7: **Ensemble strategies:** We compare accuracy with different ensemble strategies, namely *AveFL*, *OptFL* and *MixFL* (top: 10 acquisitions, bottom: 20 acquisitions). The various cases are labeled using the format 'ExFyDz', where 'E' is the epoch, 'F' is the ensemble frequency, and 'D' is the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

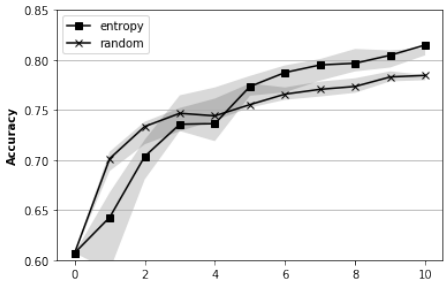
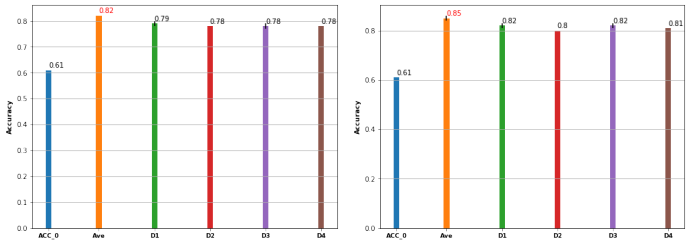


Fig. 8: Given the same number of data, AL outperforms random choice in terms of accuracy.



(a) Random (b) Entropy

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

REFERENCES

[1] Konen, Jakub, et al. "Federated learning: Strategies for improving communication efficiency." arXiv preprint arXiv:1610.05492 (2016).

[2] Konecn, J., McMahan, H.B., Ramage, D. and Richtrik, P., 2016. Federated optimization: Distributed machine learning for on-device intelligence. arXiv preprint arXiv:1610.02527.

[3] Hong, D. and Si, L., 2012, August. Mixture model with multiple centralized retrieval algorithms for result merging in federated search. In Proceedings of the 35th international ACM SIGIR conference on Research and development in information retrieval (pp. 821-830). ACM.

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

[5] Diro, A.A. and Chilamkurti, N., 2018. Distributed attack detection scheme using deep learning approach for Internet of Things. Future Generation Computer Systems, 82, pp.761-768.

[6] Lakshminarayanan, B., Pritzel, A. and Blundell, C., 2017. Simple and scalable predictive uncertainty estimation using deep ensembles. In Advances in Neural Information Processing Systems (pp. 6402-6413).

[7] Zhu, H. and Jin, Y., 2019. Multi-objective evolutionary federated learning. IEEE transactions on neural networks and learning systems.

[8] He, C., Tan, C., Tang, H., Qiu, S. and Liu, J., 2019. Central Server Free Federated Learning over Single-sided Trust Social Networks. arXiv preprint arXiv:1910.04956.

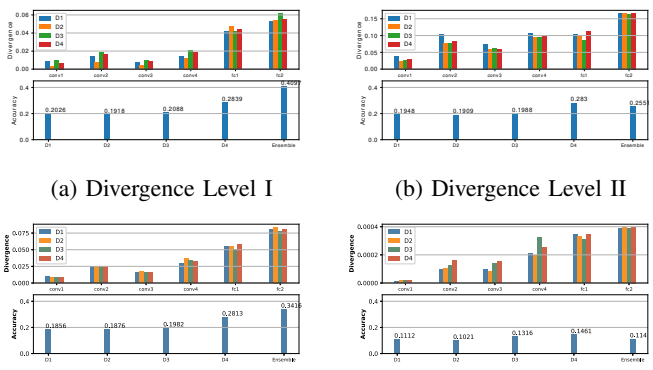
[9] Zhao, Y., Yu, C., Zhao, P. and Liu, J., 2019. Decentralized Online Learning: Take Benefits from Others' Data without Sharing Your Own to Track Global Trend. arXiv preprint arXiv:1901.10593.

[10] Torrey, L. and Shavlik, J., 2010. Transfer learning. In Handbook of research on machine learning applications and trends: algorithms, methods, and techniques (pp. 242-264). IGI Global.

[11] Qian, J., Sengupta, S. and Hansen, L.K., 2019. Active Learning Solution on Distributed Edge Computing. arXiv preprint arXiv:1906.10718.

[12] Chen, T., He, T., Benesty, M., Khotilovich, V. and Tang, Y., 2015. Xgboost: extreme gradient boosting. R package version 0.4-2, pp.1-4.

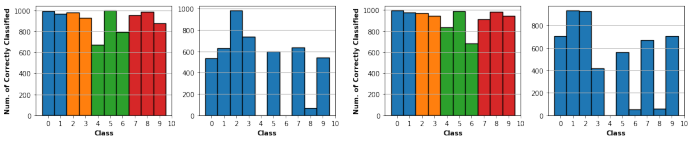
[13] Xu, Z., Huang, G., Weinberger, K.Q. and Zheng, A.X., 2014, August. Gradient boosted feature selection. In Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data



(a) Divergence Level I (b) Divergence Level II

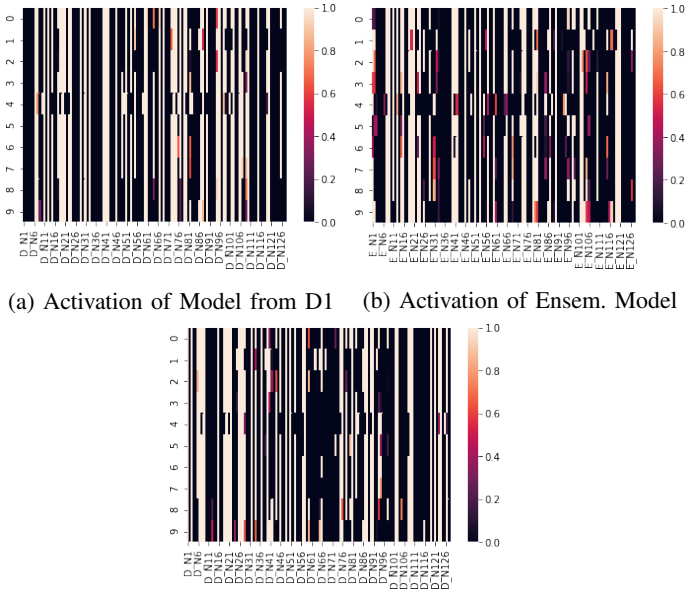
(c) Divergence Level III (d) Divergence Level IV

Fig. 10: Plot above represent the divergence and plot below is the corresponding accuracy of local models and ensemble model. X coordinator of plot (above) indicates the layers of model and bars with different colors represent the different devices. Ensemble 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 ensemble performance. Note that y coordinators are not aligned due to the different magnitudes.



(a) *AveFL* (b) *MixFL*

Fig. 11: Figure 11a is the correctly classified histogram of Average ensemble case, in the left plot the four different color bars correspond to four models of edge devices without ensemble. 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 ensemble method.



(a) Activation of Model from D1 (b) Activation of Ensem. Model

(c) Activation of Model from D3

Fig. 12: We choose 10 test images from class one and feed into local model from D1 in Figure 12a, ensemble model in Figure 12b and local model from D3 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 D1 is similar to the ensemble model, fairly different from D3 where class 1 is not generated.

mining (pp. 522-531). ACM.

[14] Klein, D.A. and Cremers, A.B., 2011, May. Boosting scalable gradient features for adaptive real-time tracking. In 2011 IEEE International Conference on Robotics and Automation (pp. 4411-4416). IEEE.

[15] Son, J., Jung, I., Park, K. and Han, B., 2015. Tracking-by-segmentation with online gradient boosting decision tree. In Proceedings of the IEEE International Conference on Computer Vision (pp. 3056-3064).

[16] Ioffe, S. and Szegedy, C., 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv:1502.03167.

[17] Pinsler, R., Gordon, J., Nalisnick, E. and Hernndez-Lobato, J.M., 2019. Bayesian batch active learning as sparse subset approximation. In Advances in Neural Information Processing Systems (pp. 6356-6367).

[18] Sarwar, S.S., Ankit, A. and Roy, K., 2017. Incremental learning in deep convolutional neural networks using partial network sharing. arXiv preprint arXiv:1712.02719.

[19] Verbeek, J.J., Vlassis, N. and Krse, B., 2003. Efficient greedy learning of Gaussian mixture models. Neural computation, 15(2), pp.469-485.

[20] LeCun, Yann, Corinna Cortes, and C. J. Burges. "MNIST handwritten digit database." *AT&T* Labs [Online]. Available: http://yann. lecun. com/exdb/mnist 2 (2010).

[21] Carey, P., 2018. Data protection: a practical guide to UK and EU law. Oxford University Press, Inc..

[22] Krogh, A. and Hertz, J.A., 1992. A simple weight decay can improve generalization. In Advances in neural information processing systems (pp. 950-957).

[23] Breiman, L., 1996. Bias, variance, and arcing classifiers. Tech. Rep. 460, Statistics Department, University of California, Berkeley, CA, USA.

[24] Zhou, Z.H., 2012. Ensemble methods: foundations and algorithms. Chapman and Hall/CRC.

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

[26] Yosinski, J., Clune, J., Nguyen, A., Fuchs, T. and Lipson, H., 2015. Understanding neural networks through deep visualization. arXiv preprint arXiv:1506.06579.