UPPSALA
UNIVERSITET

Federated Learning for Bioimage Classification

Jiarong Liang

Degree project in bioinformatics, 2020
Examensarbete i bioinformatik 30 hp till masterexamen, 2020
Biology Education Centre and Department of Information Technology, Uppsala University
Supervisor: Andreas Hellander

Abstract

Machine learning, especially deep learning, is becoming popular in biological and medical fields. A large amount of data is produced within these fields. However, due to privacy issues, data scientists are not allowed to use many of these data. A new machine learning setting called Federated learning is considered to have the ability to utilize private datasets without the leakage in privacy, providing more potential opportunities for data scientists.

In this project, we simulate federated learning under various scenarios using the CIFAR10 dataset, including model and data poisoning, client size, asynchronous node updating and sharing a public dataset. The simulation system can be edited for more scenarios as well. We also implement a federated diabetic retinopathy detection task on the cloud. We hope this work can be a reference for bioinformaticians who want to attempt federated settings in future research.

Learning without sharing

Popular Science Summary

Jiarong Liang

Nowadays, biological or medical researchers have produced a large amount of data, which seems to be a perfect support for machine learning. Machine learning, especially deep learning is becoming more and more popular in biological fields. In many cases, the researcher will ’teach’ a computer-based model to handle certain tasks using large and suitable datasets. These datasets can be selected from public databases. Many interesting attempts have already been done in applications such as developing drugs, assisting doctors, predicting epidemics, and so on.

However, in some cases, researches are hindered by lacking suitable datasets, since biological or medical records can include sensitive information and are not allowed to be shared with data scientists. For example, patients’ medical records from the hospitals should not be visited by anyone else except for the doctors. In another example, the gene sequence of the individuals are client’s genetic privacy, sequencing companies are not allowed to share them with genetic labs. In other situations where private information can be deduced from the records, sharing information can also be dangerous and illegal.

We all know that for complex tasks, feeding in more data to the model can usually result in better performance. Unfortunately, a suitable amount of data is not always available from public databases. Only large institutes have enough data to host a program, and the dataset can usually be biased. From this point of view, if we cannot share these data, the datasets from small instances become somehow ’wasted’.

A new method has been developed for avoiding such limitations in data usage. This method, which is called Federated learning, was published in 2016 and has already been used in the development of several applications such as the Google keyboard. This method can make good use of the client’s private datasets without data scientists having access to them. In this project, we will test this method’s performance under several possible scenarios, and present an example of using this technique for disease detection.

Degree project in bioinformatics, 2020
Examensarbete i bioinformatik 30 hp till masterexamen, 2020
Biology Education Centre and Department of Information Technology, Uppsala University
Supervisor: Andreas Hellander

Table of Contents

Abbreviations

CNN	Convolutional Neural Networks
DR	Diabetic Retinopathy
FedAvg	Federated Averaging Algorithm
FedSGD	Federated Stochastic Gradient Descent Algorithm
GAN	Generative Adversarial Networks
LSTM	Long Short-Term Memory
HTTP/2	Hypertext Transfer Protocol Version 2
IID	Independent and Identically Distributed
RNN	Recurrent Neural Network
RPC	Remote Procedure Call
SMC	Secure Multiparty Computation
TCP	Transmission Control Protocol

Software Links

TensorFlow	https://www.tensorflow.org/
gRPC	https://grpc.io/

Project GitHub Repository

https://github.com/Jiarong000/Thesis2020

1 Introduction

Machine learning methods have been applied to medical or biological researches for a long time. Traditional algorithms such as PCA, random forest, AdaBoost has been widely used (Benitez *et al.* 2011, Chen & Ishwaran 2012). In recent years, a subfield called deep learning is becoming popular. Traditional machine learning workflow requires manual preprocessing and feature extraction, while deep learning can learn features from raw data. (Angermueller *et al.* 2016) and (Jurtz *et al.* 2017) summarize deep learning's application in regulatory genomics, biological image and sequence analysis. Besides these, many interesting works have been done in recent years. DeepVariant aims to detect genetic variants from sequencing data (Poplin *et al.* 2018). DeepSimulator attempts to generate some more realistic base calls for simulation experiments (Li Y *et al.* 2018). Autoencoder has also been applied for denoising scRNA-seq datasets (Eraslan *et al.* 2019). Though the results of machine learning still require careful assessments from professionals in the application fields, it has a huge advantage in data mining and assisting professionals.

There is a consensus that a large dataset is required for deep learning tasks, especially when the task is complex. ChestX-ray14 is the largest dataset of chest radiographs, which contains 112120 images from 30805 unique patients. CheXNeXt is trained over this dataset and claims to approximate expert's performance (Rajpurkar *et al.* 2018). In another research, several structurally distinct antibacterial molecules have been discovered with the assistance of a model trained with an empirical dataset of size 2335 (Stokes *et al.* 2020). In many other cases, a suitable dataset can be collected from biological databases such as NCBI and EMBL or other specific databases such as TCGA and ICGC.

However, it is not always possible to collect an appropriate dataset. In many cases, datasets are small and unbalanced. In other cases, annotations are not suitable for the specific training task. Many institutes, for example, hospitals or sequencing companies, own a private dataset of their clients and have the ability to perform annotation. Due to privacy concerns and restrictions, this part of data is usually not allowed to be shared with other institutes or to the public, resulting in many small and isolated datasets.

A new machine learning setting called Federated learning is considered to have the potential of utilizing isolated datasets without leakage in personal privacy. In this project, we test federated learning's behavior under various scenarios and perform an example federated medical image classification task on the cloud. We hope this project can be a friendly reference for understanding several possible issues in building up a federated system, encouraging more federated attempts in medical or biological researches.

We mainly research issues in the application in this project. Communication or security problems will not be focused on.

2 Background

2.1 Federated Learning

Federated learning is a machine learning strategy where many clients can collaboratively train a model when their local dataset is inaccessible to each other. The system will be orchestrated by a central server, in which training results from clients are aggregated. Compared to centralized learning, it requires less storage or computational resources in the central server and most importantly, protects each client's private data.

There are many domains in federated learning. Cross-device federated learning is implemented on a large number of small devices while cross-silo learning is for collaboration among institutions (Kairouz *et al.* 2019). From another aspect, in most training cases there will be horizontal federated learning where client datasets are partitioned by samples and share similar feature space. In this case, gradients or weights are aggregated. On the other hand, client datasets only share user space in vertical learning, in which intermediate results such as hidden layer's representations are shared(Liu *et al.* 2018). Federated transfer-learning, applied when the intersection in both feature and user space is small, is also possible (Liu *et al.* 2018).

A typical federated training mainly contains several basic steps (Kairouz *et al.* 2019). In the simplest federated model, the server will first select clients who meet the requirement and broadcast the model to them. Secondly, clients will train the model with their local dataset and upload the results (i.e. weights, gradients, intermediate layers) to the server. Then the server can aggregate the information and perform the evaluation. These steps will be repeated until the aggregated model pass the evaluation. Additional techniques and methods are applied to improve training performance or security, which will be discussed in later sections.

2.2 Federated Algorithms

Federated Averaging (FedAvg) and Federated SGD (FedSGD) are two popular algorithms in federated learning and many optimization algorithms are developed from them. In FedAvg settings, client weights will be collected and averaged while in FedSGD settings, gradients will be collected (Mcmahan *et al.* 2016).

Many other optimization algorithms are developed for various purposes, (Kairouz *et al.* 2019) provides a summary of popular ones, such as FedProx (Li T *et al.* 2018) and SCAFFOLD (Karimireddy *et al.* 2019), using either proximal term or control variates to reduce the effects caused by heterogeneous clients. This means, for example, model weights are updated towards minimizing the loss function (f) in FedAvg. In FedProx, weights are updated to minimize ($f + u||w-W^t||^2$) in which a proximal term that represents the distance between client weights (w) and global weights (W^t) is added in as a restriction factor. Other methods such as

federated transfer learning (Liu *et al.* 2018) share hidden elements instead of gradients or weights. (Peng *et al.* 2019) uses the idea of GAN and a dynamic attention mechanism to align the learned representations from source nodes to the target nodes for knowledge transfer.

Below is the pseudo-code of FedAvg, which is used in this project. In each client node, gradient descent is used to minimize the loss function f, where η is the learning rate to control the speed of updates. The server collects updated weights from each client and generates the global weights for that round via averaging, weighted by the node size.

Server executes:
  Initialize w_0
  For each round t = 1, 2,..., T **do**
    For each client index by k= 1, 2,..., K **in parallel do**
      $w_{t+1}^k \leftarrow$ ClientUpdate(k,w_t)
    $w_{t+1} \leftarrow \sum_{k=1}^{K} \frac{n_k}{n} w_{t+1}^k$
ClientUpdate(k,W):
  $B_k \leftarrow$ (local dataset of the k^{th} client in batches)
  For local step s = 1,..., S **do**
    For batch b $\in B_k$ **do**
      $W \leftarrow W - \eta \nabla f(W,b)$
  Return W to server

Algorithm 1. FedAvg

2.3 Previous Work

Many pieces of research are implemented on current challenges in federated learning includes communication cost, security protection, statistical and system heterogeneity (Li T *et al.* 2019b). The communication cost is usually a bottleneck for cross-device learning. To save the cost, compression schemes and multiple local updating can be introduced to the settings (Li T *et al.* 2019b). Privacy issues, such as retrieving information from two successive models, have solutions such as SMC, differential privacy and homomorphic encryption. These solutions have already been used in popular federated frameworks that support both deep learning and traditional machine learning. Other techniques such as blockchain, can also be integrated to make the system more secure (Passeratpalmbach *et al.* 2019).

Statistical heterogeneity (such as different label distributions among nodes) can also cause performance degradation. Several methods to solve this issue aim to make the local training set less skewed, including sharing a small set of data (Zhao *et al.* 2018) and augmenting local set towards IID (Jeong *et al.* 2018). Methods like PFNM (Yurochkin *et al.* 2019) and FedMA (Wang *et al.* 2020) match hidden elements with similar feature extraction signatures, since FedAvg has decreased in accuracy and is slow due to the permutation invariance of neural network parameters (i.e. for any given network, many variants of it only differ in the ordering

of parameters). Some other methods, such as FedProx (Li T *et al.* 2018) and RSA(Li L *et al.* 2018) add restrict between local and global parameters to optimization function. FedNAS (He Chaoyang *et al.* 2020) is an automating federated learning method that claims to outperform manually settled architecture on non-IID datasets. Fairness is also an issue, while the system can be biased toward some clients, q-FFL aggregates the weights in the form where clients with higher loss are weighted more (Li T *et al.* 2019c).

System heterogeneity is also common. In some scenarios, such as in cross-device learning, since it is not a usual case to wait for all devices to be ready for training, the system will train on only the available devices at each time. FedCS (Nishio & Yonetani 2019) is a strategy for managing federated learning among heterogeneous devices based on their resource information. In other cases, nodes will not update in synchronous ways if some nodes are too slow to be waited for, or nodes can be added in or drop out from the system during the training process. These are compromises for real-world applications and probably can cause the system to act in slower ways than identical. However, we are still uncertain about the extent of this impact.

Another issue in federated learning is the robustness to attacks. There are targeted and untargeted attacks, in which untargeted attacks aim to reduce global model's accuracy while targeted attacks aim to perform an incorrect mapping on a fraction of samples and keep other mappings correct. There are several widely used methods to solve this issue (Fang *et al.* 2019): Krum select the local model which is the most similar to the global model, Bulyan select several models and average them, Trimmed Mean remove extreme values of each parameter before averaging them, Median select the median of each parameter. Since malicious messages can hide in the noisy updates in a heterogeneous system, SAGA (add up the average of the previously-stored gradient of each sample) and Byrd-SAGA (the geometric median version of SAGA) reduce the variance of benign workers' gradients to enhance robustness to attackers (Wu *et al.* 2019), while another research (Suresh et al. 2019) suggest adding a small amount of noise (i.e. weak differential privacy). FedProx (Li T *et al.* 2018) and RSA(Li L *et al.* 2018) can also enhance robustness to attacks.

The previously mentioned optimization methods can improve the performance of federated learning under certain circumstances, however, in many cases, the optimized performance in federated settings is still worse than that in centralized settings. In some other cases, the new algorithms may not suitable for the training task. For example, PFNM (Yurochkin *et al.* 2019) does not support CNN and RNN while FedMA (Wang *et al.* 2020) supports CNN and RNN of simple architectures. When coming to issues of attacks, common methods like Krum, Bulyan, Trimmed Mean and Median are proved to can be avoided if the defense methods are known (Fang *et al.* 2019). The targeted attacking strategy DBA (Xie *et al.* 2020) distributes the attack into pieces, making it mild and harder to be detected, and aggregates it along with the model. These examples indicate that issues of heterogeneity and attackers are still challenging.

Surprisingly, even when multiple types of research in optimization methods are available, we cannot summarize an overview of these issues' impact on federated learning. The impacts are tested mostly in limited scenarios as a comparison to the optimization methods. In different researches, the author may also use different datasets and evaluation standards, making the comparisons hard. In (Kairouz *et al.* 2019) the authors purpose the mathematical methods to calculate convergence rated for non-IID datasets, however, only for ideal cases and is just theoretical. This fact can let many people stay unsure about the extent of each issue's impact on their federated system, making it hard to decide which aspect of the system should be firstly optimized. This can be a problem, especially when the available optimization methods are not suitable for the system, or are very costly, or cannot be integrated. In this project, we test several common issues in federated learning (specifically, heterogeneity and untargeted attack) with different amounts of data in different distributions. We hope the result can be a clear and easy-to-understand reference for those who attempt federated settings in the future.

2.4 Federated Frameworks

In the Appendix of (Kairouz *et al.* 2019), several popular federated frameworks are summarized. TensorFlow Federated and PySyft provide tools for federated learning based on the widely used TensorFlow and PyTorch software. A platform called PyGrid is also under development, using PySyft as its basic library. However, by the end of May 2020, their official versions have not yet been released, only simulation functions are available.

Another software called Federated AI Technology Enabler (FATE) has been released in 2019. FATE already have enabled several functions in both traditional learning and deep learning. Meanwhile, more functions are under construction.

In federated frameworks, clients need to communicate with the server. Browsing their source code, TensorFlow Federated seems to use gRPC and an example in PySyft uses WebSocket. gRPC is a high-performance RPC framework with HTTP/2-based transport. WebSocket provides computer communication over a single TCP connection. In this project, we use gRPC for communication.

2.5 Future in Biological Fields

A successful case of brain tumor segmentation in federated settings has been reported in 2018 (Sheller *et al.* 2018). However, this work is done by computer scientists, not medical researchers. Federated learning contains interdisciplinary problems from computer science fields, sometimes can be too complex from non-experts.

But we aware that many drug designers are now using deep learning techniques, which also sound like high-end computer science techniques a few years ago. The machine learning tools like TensorFlow and PyTorch are becoming sophisticated and convenient, and are becoming much friendly to the beginners. Another field called automated machine learning aims to

assist non-experts to apply machine learning techniques in many aspects (Yao *et al.* 2018). Google's AutoML (Real *et al.* 2020), AutoAugment (Cubuk *et al.* 2018), Swish (Ramachandran *et al.* 2017) are popular examples.

At the same time, federated frameworks such as FATE can provide secure federating experience for their users. There are already attempts to apply AutoML in a federated way, typically the federated Neural Architecture Search (He Chaoyang *et al.* 2020, Xu *et al.* 2020). In the foreseeable future, collaboratively training a model with data from different institutes might no longer be a difficult task for biological researchers.

3 Numerical Experiments

In this project, CIFAR-10 dataset (Krizhevsky 2012) has been used for locally simulating different scenarios of federated learning. The model used in the experiments is provided by Mattias Åkesson from Scaleout Systems. The training design is not the state of the art, but it is sufficient for testing scenarios for federated learning without loss of generality. The code is written in Python3 and TensorFlow 2.0.1 is used for implementing machine learning tasks.

3.1 CIFAR-10 dataset

CIFAR-10 dataset consists of 50,000 training images and 10,000 testing images of 10 balanced classes, each image is in color format (i.e. with 3 channels) and the size is 32x32 pixels. The 10 classes are airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck (labeled using 10 integers: 0-9). Image samples and more details can be found on the official website (http://www.cs.toronto.edu/~kriz/cifar.html).

In this project, we use all testing images as our testing set and subsets of training images as training sets. Our training sets have 2 different sizes, that is 4000 or 40000 samples, we will label them as '4k dataset' and '40k dataset' in the following parts. The training sets are randomly selected from 50,000 training images.

The training sets are divided into pieces (which could be viewed as clients). The divisions are implemented in a manner that simulates IID and non-IID circumstances. In the Appendix, there is an example of dividing 4k dataset to 10 workers under 2 class non-IID circumstances. In this case, ' 2 class non-IID' means each client contains samples from 2 classes, in which the sample number for classes are equal. For IID circumstances, each client contains samples from 10 classes. Mention that though the sample distributions in nodes can be skewed, samples in the total set (i.e. the 4k and 40k dataset) are balanced.

According to the results of pilot tests, tests for each scenario are implemented using the 4k dataset under 2 class, 5 class and IID circumstances or using the 40k dataset under 2 class, 3

class, 5 class and IID circumstances. If not specifically mentioned, we will simulate all these distributions for each scenario.

3.2 Model and Simulation Designs

The model for simulation tasks is a CNN model, input images from CIFAR-10 dataset and classifies them into 10 categories. Figure 1 shows its architecture. During the training process, ADAM optimizer is applied with the default learning rate (1e-3); the loss is calculated by categorical cross-entropy. The altering of the model structure is also possible if input and output format stays constant, but here we only use the provided model.

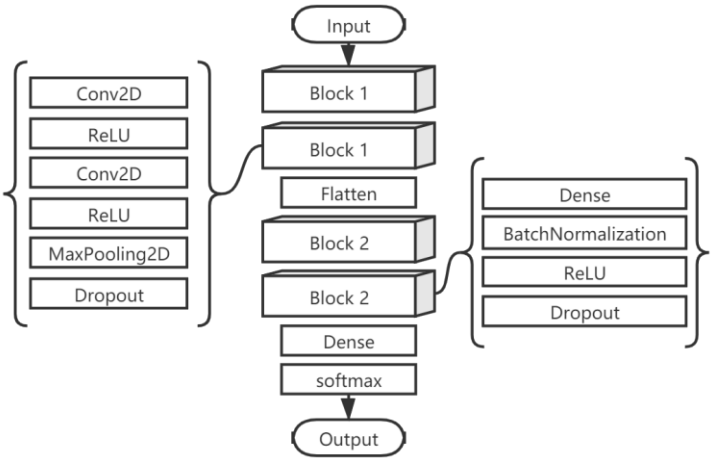


Figure 1. CNN model for numerical experiments.

The simulation system is a virtual federated system, which includes one central node and several client nodes. In this case, we generate files of indexes for samples as virtual client nodes before the training starts. When the training starts, the system will initialize a model with randomized weights (this can be done automatically by the TensorFlow) and save it in a folder that is assumed to be the virtual central node. During the training process, for each index file, the system will load model weights from the central node, apply training with indexed samples and cache the resulting model parameters (also save the training records for each index file). After that, the cached local models are aggregated to form a new global model (using FedAvg mentioned in the previous section). This new model is then evaluated by the testing set and saved for the next epoch. Figure 2 briefly shows this process.

Currently, the system only supports simulation with CIFAR-10 dataset, and users need to run a separate script to generate index files. The scrip will decide the number of nodes to be generated, the size of the node and the skewness of the distribution (i.e. the extent of IID and non-IID).

Users can control several parameters in the system, here is their default settings: Number of global rounds (no default setting), Number of local rounds (set as 1), Early stop condition (no default setting), Augmentation (set as False), Number of unusual nodes ('delayed node', 'poisoning node'; both set as 0), Samples to be shared with the public (set as 0). When testing different scenarios, we will change the parameter accordingly, and keep other parameters in default settings.

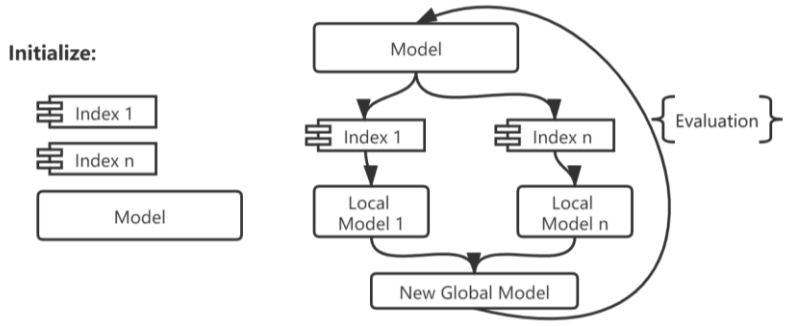


Figure 2. Simulation process.

3.3 Pilot Tests

In pilot tests, we test the system from two aspects. From the first aspect, we perform the test under IID circumstances and use datasets of different sizes (i.e. 500, 600, 700, 800, 900, 1000, 1500, 2000, 3000, 4000, 8000, 40000). From the second aspect, we perform the test with the 4k dataset under various non-IID or IID circumstances (i.e. 1-9 class non-IID and IID). The datasets are divided into 10 clients.

3.4 Poisoning

When testing model poisoning, the datasets are divided into 10 clients and add in an anomaly node that returns arbitrary weights in each period, performing untargeted attacks. Since TensorFlow will randomly generalize the weights when initializing the model, the attack can be performed by initializing a model with the same architecture and return its weights during each updating. The declared size of the anomaly node is also controlled, we label it as a certain proportion of the total training set. This proportion is 1% or 10% when using the 4k dataset, and is 1%, 3% or 10% when using the 40k dataset.

In previous tests, we assume the period of poisoning is one global epoch. In other words, poisoning frequency is once per global epoch. To check the influence of frequency, we change the frequency to once per 3 global epochs when node size is 3%, and to once per 10 global epochs when node size is 10%. These tests are performed only with the 40k dataset under IID condition.

We also test data poisoning with the 40k dataset under IID condition. There are 10 clients and one anomaly of size 1%, poisoning frequency is once per global epoch. The attack can be achieved by training the anomaly node with the dataset in which all labels are randomly shuffled.

3.5 Client Size

In some tests, we distribute our data to 10 clients, in other tests to 40 clients. This raises concerns about the effect caused by the client size. In this scenario, we compare the training performance between splitting the same amount of data to 10 clients and 40 client cases, using the default system.

3.6 Delayed Update

In this scenario, we assume some nodes will update at slower speeds than the others. The training sets are divided into 40 clients and the situation is tested from 2 aspects.

From the first aspect, we test for various proportions of delayed nodes. The proportions of delayed nodes (we call it ' delay proportion' in later texts) for 4k datasets are 20%, 50% and 80%, for 40k datasets are 25%, 50% and 75%. The numbers have no exact meaning, they are summarized to categories '<50', '50', and '>50'. Meanwhile, the default system will be labeled to category '0', since it has no delayed node. In these tests, the delayed nodes are 3 times slower than the others. In other words, 'delay speed' equals 3. To be more specific, the delayed nodes will load the weights at global round T, and upload the weights at global round T+3. New weights won't be loaded until current weights are uploaded.

From the second aspect, we test for various speeds of delayed nodes. The tested speeds are '3', '12' and '30'. Also, the default system will be labeled as '0'. In these tests, ' delay proportion' are in category '<50'.

3.7 Sharing and Resampling Data

Logically we understand that 1 class non-IID never behaves better than random since locally there will be almost no effective update. In pilot tests, other seriously non-IID datasets also show a certain decrease in performance. Previous research has suggested a data-sharing strategy that can improve training performance when using 1 class non-IID data (Zhao *et al.* 2018) if a small IID set can be published for warming up the model or sharing between nodes. Another attempt augments local set toward IID (Jeong *et al.* 2018).

In this scenario, we attempt on 3 different methods of using the shared data. Since oversampling methods we used here are too costly, we only test this scenario with 4k dataset under 1 class and 5 class non-IID circumstances for 200 rounds, with '2.5%' sharing amount.

Using this strategy, after receiving the shared data, each client node will contain a mixed sample pool of shared and local data. Method 1 is the direct usage of the entire pool. Method 2 will under-sample a balanced subset from the pool, and Method 3 will oversample the pool.

Method 2 can be achieved using 'RandomUnderSampler' from the 'imblearn' package. This function can randomly pick samples from each class to generate a smaller balance set. In other words, after applying this function to the data pool, we will get a balanced dataset in which the size of each class equals to the size of the smallest class.

Method 2 can be achieved using either 'SMOTENC' or 'ADASYN' from the ' imblearn' package. These two functions can synthesize new minority samples from the current ones. The idea of SMOTE is to randomly pick up two nearby minority samples within the group, and generate the new sample between them. ADASYN will calculate the distribution of minority samples, and generate new samples based on the distribution. In our settings, after oversampling, we will get a balanced dataset in which the size of each class equals to the size of the largest class.

One thing has to be mentioned that the above functions support the calculation only for vectors, but our inputs are image matrix. Thus we flatten the images to vectors, apply to sample, and then turn it back to matrix accordingly.

4 Cloud Implementation

In this project, OIA-DDR dataset (Li T *et al.* 2019a) has been used for implementing an example medical image classification task on the cloud. Also, we use Python3 and TensorFlow 2.0.1 for machine learning tasks, and gRPC is used for communication.

4.1 OIA-DDR dataset

The OIA-DDR dataset provides high-quality diabetic retinopathy (DR) images and annotations. The grading annotation labels the images to six classes: no DR, mild, moderate, severe, proliferative and ungradable. In this project, we neglect images from the 'ungradable' class, using 6266 images from 'no DR' class as healthy samples and 6256 images from the rest 4 classes as DR samples.

Images in this dataset are in different sizes and qualities. We preprocess the images using the methods from Kaggle's public kernels, crop and resize them to 224x224 pixels. We attach referenced kernels' links and several processed images in the Appendix.

We select a balanced training set of 10000 samples, use the rest 2521 samples as the testing set, and generate both IID and non-IID version of the training set. In the IID version of division, there are 3 clients of equal size, each contains a balanced subset. In the non-IID

version of division, there are a large and balanced client and 2 small and unbalanced clients. Here, balance refers to the balance between 'no DR' samples and 'DR' samples. Figure 3 shows the original OIA-DDR dataset and the divided training set of the non-IID version.

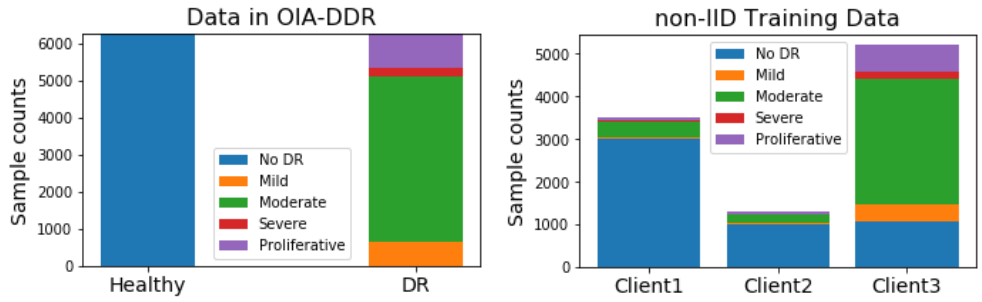


Figure 3. Sample distribution. *Left*: Samples in OIA-DDR dataset. *Right*: Sample distribution in the non-IID training set.

4.2 Model and System Designs

The model used in cloud implementation is modified from the VGG16 network and initialized from model weights that are pre-trained on ImageNet database, most layers are set as non-trainable. Inputs are images from the preprocessed OIA-DDR dataset, and the outputs are classifications to 2 categories. Figure 4 shows its architecture. ADAM optimizer is applied with the learning rate (*lr*), initialized at 1e-4 and decayed by rate 0.05 in every global epoch; the loss will be calculated by binary cross-entropy.

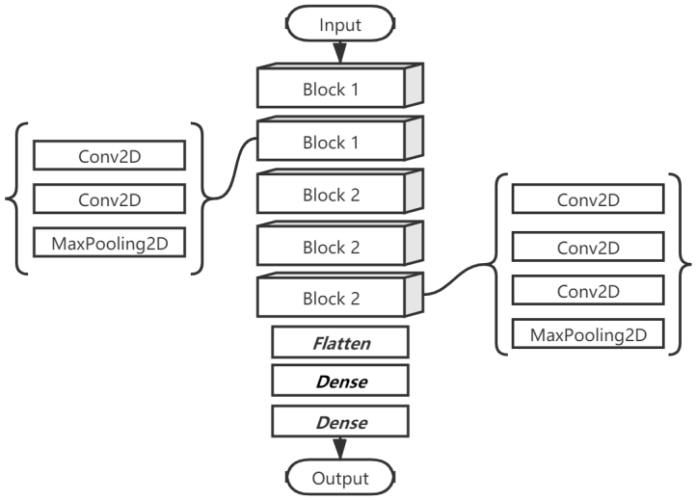


Figure 4. Model for cloud implementation. Trainable laters (2 Dense layers) are labeled in italic. Other layers keep using the pre-trained weights on ImageNet.

The function of the learning rate is:

$$lr = \frac{1e-4}{1 + 0.05 * epoch}$$

The system on the cloud consists of 1 central server and 3 clients, using gRPC for communication. The system is synchronous, which means the server will wait for all clients to be ready. Figure 5 shows the processes in the system. Assume the system has information on the client number. Mention that this system is very simple, mechanisms to prevent failures or enhance security are not added in.

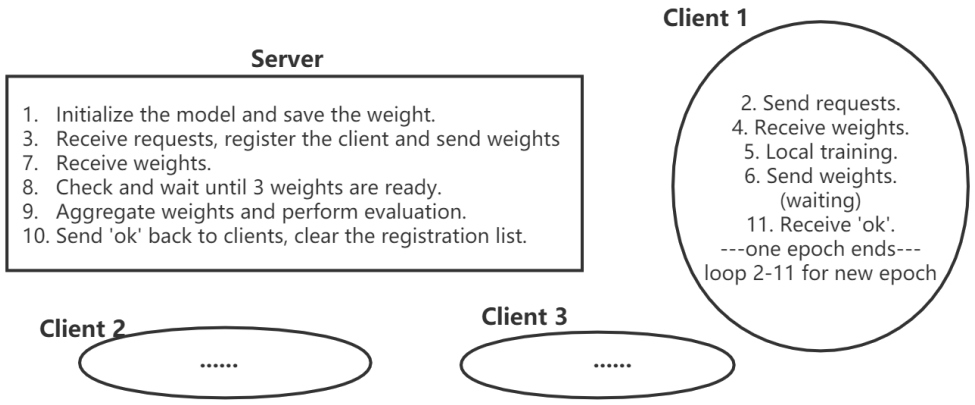


Figure 5. Cloud System.

5 Results

This part includes the results for both numerical tests and cloud implementation.

In an ideal training process, the best accuracy and minimal loss will be achieved at the convergence epoch. In our training during numerical experiments, we have observed a slight increase in accuracy after the loss reaches to its least, in which case the model starts to overfit the training data. Since our loss function is cross-entropy, this situation is possible, for example, when some samples are classified to more deviated predictions from correct results while slightly deviated predictions are corrected. In such cases, it is hard to tell if an increase in accuracy can represent the increase in performance. Also, other better models might be able to avoid this situation. Thus, we assume our model performs best at the global round with minimal loss, the accuracy at that round represents the model's best performance. In our following plots, we will label this point by a colored square.

Since the original results have fluctuations, we show and manipulate on windowed testing accuracy and loss in this report, with window size equals to 21. In this section, we show statistical summaries or part of our testing results. The full and original records are available in the GitHub repository, including the accuracy and loss curve for both training and testing datasets. Also, we will append all windowed results at the end of the report.

5.1 Pilot Tests

Figure 6 compares centralized and federated learning's behavior under IID circumstances, using training sets of different sizes. It is obvious that the performance of both federated and centralized learning has been improved when the size of training sets become larger, and testings in federated settings always converge at a slower speed than in centralized settings. The accuracy at the 500th epoch of federated and centralized learning has a small difference, probably because the tested datasets are IID distributed.

Figure 6 compares centralized and federated learning's behavior under various non-IID or IID circumstances, performed with 4k datasets. It is obvious that accuracy and convergence speed will decrease if the degree of non-IID increases.

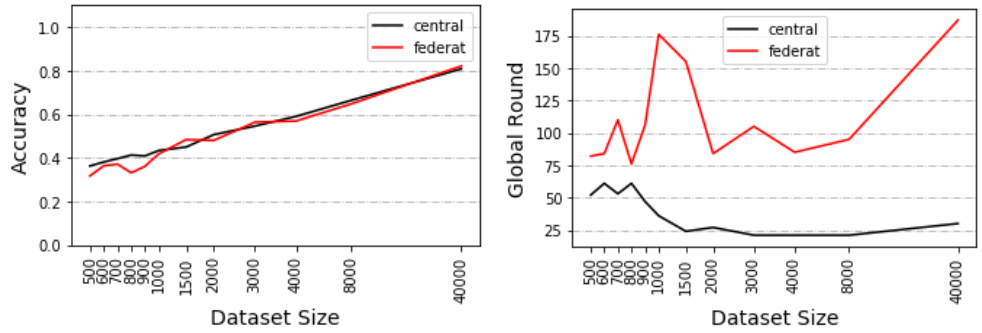


Figure 6. Pilot tests comparing various dataset size, performed under IID circumstance. *Left*: Accuracy at minimal loss. *Right*: Round index at the minimal loss.

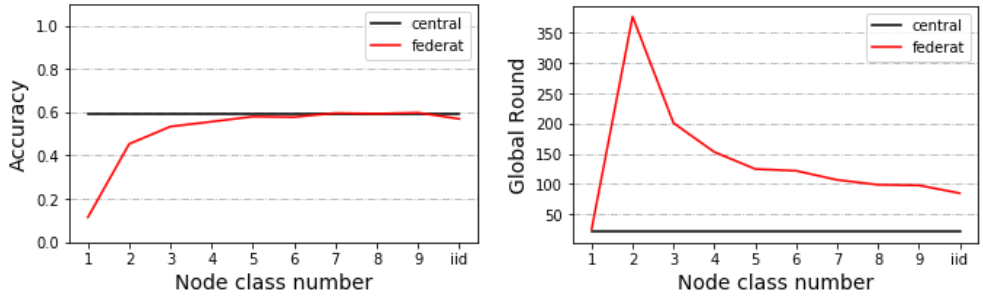


Figure 7. Pilot tests comparing various IID or non-IID circumstances, performed with 4k dataset. *Left*: Accuracy at minimal loss. *Right*: Round index at minimal loss.

We also have observed an unusual trend when testing between 4k and 40k datasets under 2 class non-IID circumstances. In other tests, training with the larger dataset always results in higher accuracy, while in this test the trend is inverted. According to our results, this inverted trend is also detectable in other scenarios under 2 class non-IID circumstances. Currently, we cannot explain this issue.

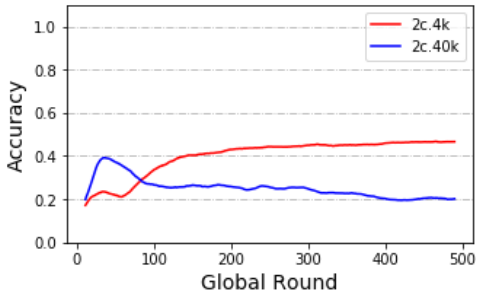


Figure 8. Test record of training with 4k and 40k dataset under 2 class non-IID circumstance.

5.2 Poisoning

In this section, we use ‘p%-f’ to represent that the system is affected by the anomaly node of size p% at every f epochs. If using ’p%’, it is equal to ’p%-1’.

The presence of an anomaly node leads to performance degradation. The influence is related to both the size of the anomaly node and the poisoning frequency. Comparison among ‘1%-1’, ‘3%-1’ and ‘10%-1’ shows the influence of node size, larger anomaly node can cause much serious influence. On the other hand, the difference between ’3%-1’ and ’3%-3’, or between ’10%-1’ and ’10%-10’ shows the influence of poisoning frequency. Interestingly, similar accuracy trends among the situation ‘1%-1’, ‘3%-3’ and ‘10%-10’ have also been observed, they have the same average anomaly node size per round.

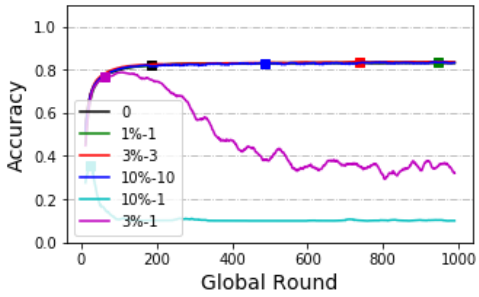


Figure 9. Accuracy curve of model poisoning tested with 40k dataset under IID circumstance.

Models trained with large dataset seems to have a higher tolerance to attacks from anomaly nodes. In the case of the 1% size anomaly node, we cannot observe performance degradation in model trained with an IID 40k dataset, while this degradation is obvious in models trained with 4k dataset. On the other hand, no obvious decrease pattern among non-IID groups has been observed. Though we have observed a slight decrease in some non-IID groups (while there is no decrease in the others), the decrease pattern is still unclear.

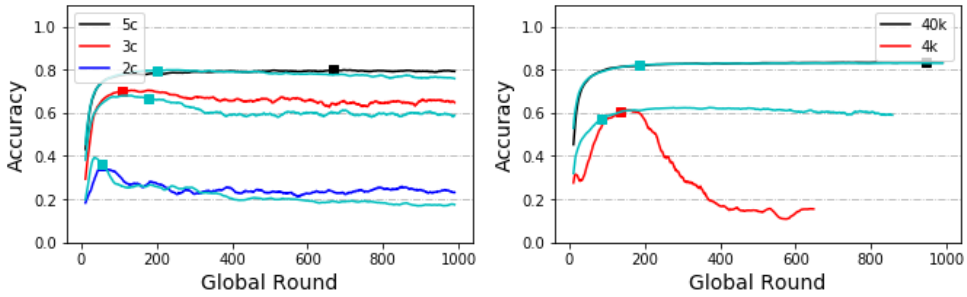


Figure 10. Accuracy curve of model poisoning. The light blue line without any label is the original curve without any poisoning for each case. *Left*: Testing with 40k dataset and '1%' anomaly node. *Right*: Testing under IID circumstance and '1%' anomaly node.

From the result of data poisoning, we can see the influence is much weaker than model poisoning attacks. In Figure 11, we can see the '3%' model poisoning attack results in an obvious degradation while '3%' data poisoning attack only shows a slight influence.

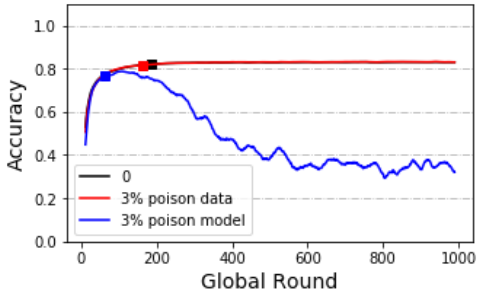


Figure 11. Accuracy curve of data poisoning and model poisoning. Testing with 40k dataset under IID condition and '3%' anomaly node.

5.3 Client Size

From Figure 8 below we can see training with 40 client nodes is slower than training with 10 client nodes while the difference between their accuracy is not obvious. Statistical tests are applied and the results show a significant difference between convergence speeds of the two groups (p-value=0.0059). Differences in accuracy are not significant (p-value=0.71).

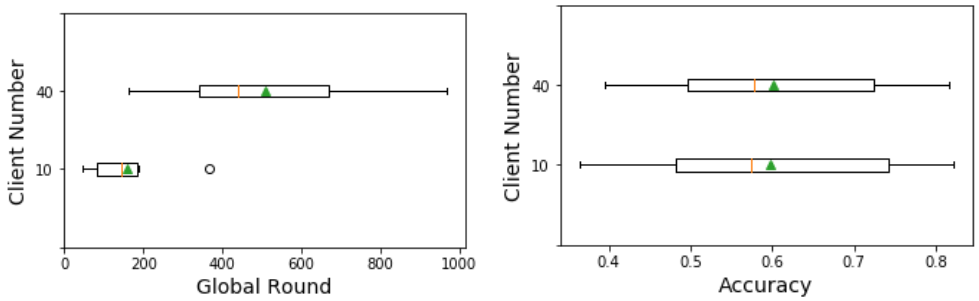


Figure 12. Box plot of the round index (*Left*) and accuracy (*Right*) where loss is the minimum, group by client number.

5.4 Delayed Update

The first part of this scenario compares the results from different proportions. When applying statistical tests, we group results when delay speed is 3 to categories '<50', '50', and '>50' regardless of other information (i.e. dataset, IID or not), and use the result set from section 5.3 as category '0' (using only the results of 40 clients). In the second part, we compare the results of different delay speed, grouping results to '3', '12' and '30'.

Statistical tests show a significant difference in convergence speed between with and without delay (p-value=0.016) when comparing between '0' and ' <50' (proportion) or '0' and ' 3' (speed). Surprisingly, this significance disappears as delay speed and proportion increases. Significance is also detected between speed group '3' and '12', showing a possible recovery when delay speed became larger. On the other hand, there is also no significance between proportion group '<50' and '50' or '50' and '>50', making the effect of delay node proportion on convergence speed ambiguous.

Delay proportion and delay speed might also influence accuracy. Statistical results show a significant difference between group '0' and proportion groups larger than 50 (p-value=0.01 and 0.005). On the other hand, changes in delay speed have no obvious influence over accuracy.

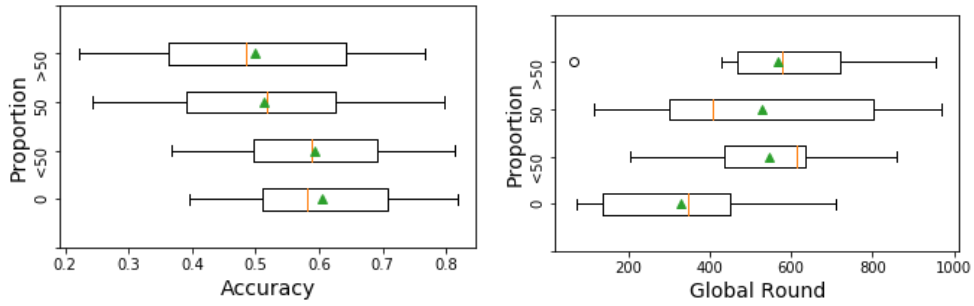


Figure 13. Box plot of the round index (*Left*) and accuracy (*Right*) where loss is the minimum, group by the proportion range of delayed nodes when delay speed set to 3.

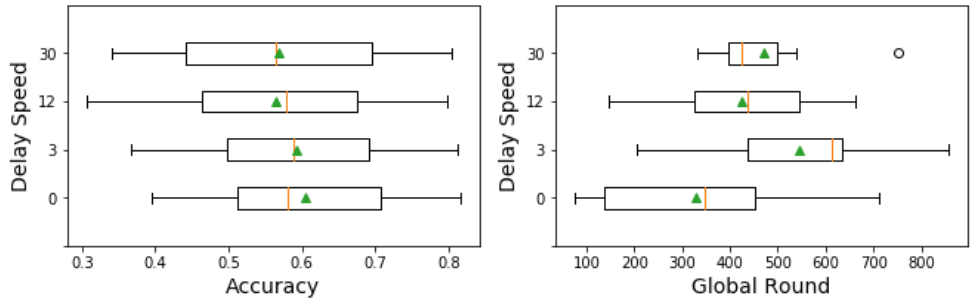


Figure 14. Box plot of the round index (*Left*) and accuracy (*Right*) where loss is minimum, group by delay speed, when delay proportion ranges in '<50'.

5.5 Sharing and Resampling Data

Figure 11 shows the results between directly using the mixed dataset and doing resample on the mixed dataset, tested with 4k dataset under 1 class and 5 class non-IID. Oversampling strategies seem to converge at a faster speed than the share-only method and under-sampling method. When data is seriously non-IID (the 1 class cases), all those methods perform better than without sharing but still cannot reach the accuracy level of the IID case. While the data is not seriously non-IID (the 5 class cases), share-only and oversampling methods only have subtle influences on the accuracy curve, while the under-sampling method seems to have placed a negative influence on original accuracy.

Also, the oversampling methods ADASYN and SMOTENC require a large amount of computing. In this case, since the share-only method can be much convenient and have similar or even better results than the resampling ones, this method, this method performs best for our system.

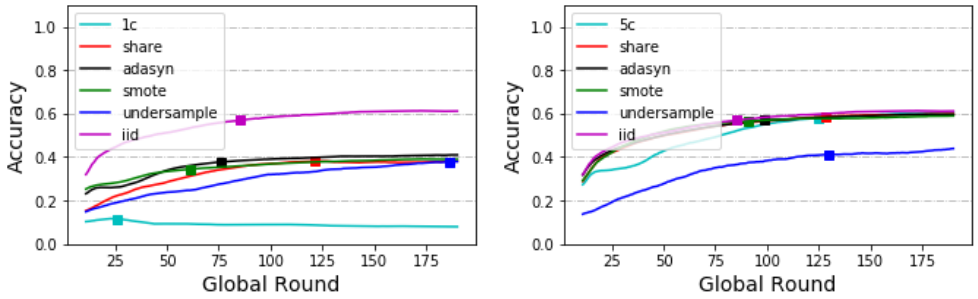


Figure 15. Accuracy curve of various sharing and resampling strategies, testing with 4k training set and 2.5 % shared set. *Left*: Results for 1 class non-IID circumstance. *Right*: Results for 5 class non-IID circumstances.

5.6 Cloud implementation

We have successfully performed our example task on the cloud. Figure 16 compares the accuracy curve for centralized training, IID and non-IID federated training, and training with the largest subset. In this figure Y-axis is zoomed to show a much clearer difference, the results in the original scale are available in the Appendix. Similar as in the numerical experiments, federated training performs worse than the centralized ones but still better than training with the subset. In this case, since the training is only weakly non-IID, the results of IID and non-IID trials are similar.

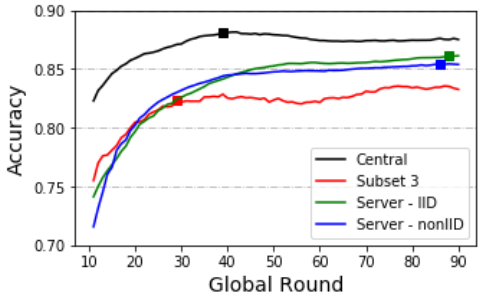


Figure 16. Accuracy curve of implementations with the OIA-DDR dataset.

6 Discussion

From previous researches, we already know that model or data poisoning can result in performance reduction and many defense methods have been suggested for this issue (Kairouz *et al.* 2019). But we have not found a test to compare this effect under different sizes of dataset and distributions. In our testing, we have observed that model poisoned by the same average strength per round ('1%-1', '3%-3' and '10%-10') have similar behavior. We have also observed a stronger resistance to model poisoning when training with a larger dataset and data poisoning will result in a weaker effect.

In this project, we perform the simplest untargeted poisoning in repeated form. Another research tests targeted poisoning in both single-shot and repeated form (Bagdasaryan *et al.* 2018). In this research, the author performs stronger attacks using the idea of model replacement, some of the single-shot attacks will gradually be forgotten, but some attacks can be more durable than others. Another paper found backdoors in pre-trained networks can be inherited (Gu *et al.* 2019). Also, if several nodes are being controlled by the attacker, they can collaboratively perform the attack in a way that is difficult to detect (Xie *et al.* 2020). Many other attacking methods can also be efficient and flexible enough to escape current defense methods. Poisoning remains a problem, though in our testings the system is comparatively less affected in slight or low-frequency attacks.

When testing with the delayed update, the system acts slowly when the delay happened. This decrease in speed seems to be reduced when delay speed increases. A possible explanation of this is that delayed nodes of larger speed upload their weights at a lower frequency than others. Due to the ambiguous statistical results, we are still uncertain about the larger delay proportion's effect on speed to reach minimal loss. The situation of '>50' can be viewed as a small proportion of nodes is acting at a faster speed than the others, while the rest of the system is three times slower than the original one. Logically, the overall system should act slower, but we cannot make a conclusion from our current results.

We have observed a decrease in accuracy when delay proportion is equal to or larger than 50%. The raise in delay speed does not obviously influence the accuracy, possibly because the proportion of delayed node of this test is small. We have prepared two conjectures for this. Firstly, considering the uploading frequency, we expect accuracy to be less affected when delay speed is increased since the weights from the delayed nodes can be viewed as slightly poisoned nodes (which is the reason of decrease in accuracy). From the other side, since data in the delayed part is not well trained, the system has less effective training data than usual, accuracy should be lower than usual. These two conjectures need further testing. To summarise, our current results suggest that the majority of nodes should be synchronized when designing the federated system.

In the test with client size, we detect no decrease in accuracy, thereby increasing the confidence of inviting small clients to the system. However, we are only testing the cross-silo cases, where clients still hold a certain amount of local dataset and will not drop out. In cross-device cases, in the most extreme cases, there could be millions of clients and each client is a sample. In such cases, many other aspects need to be considered. Also, in our experiments, we assume all clients can afford computation resources for training. There could be a case where clients cannot provide surplus computing resources for training. In such cases, hiring cloud resources still face privacy issues. Techniques like pruning can save resources, but we are unsure about its effect in federated settings.

When testing with shared data, the performance can only be improved when the training data is seriously non-IID. In our cases, the share-only strategy seems to have similar or better performance than the resampling methods and can be less resource consuming. Current methods we used for oversampling fail to improve the share-only method, possibly because these methods are not specific for generating images. ADASYN and SMOTE generate new images pixel by pixel, using the information from the original samples. This is not a suitable method for generating images, since objects in original images may have different colors, shapes, positions, and so on. There are suitable methods for image augmentation, but augmentation can also improve the system without oversampling. Here we only prove that even unsuitable oversampling methods have little negative influence. However, using the share-only method and augment the whole dataset can be effective and a much convenient option.

Except for sharing the global data, as mentioned in the background section, there are other methods to solve data heterogeneity. FedMA (Wang *et al.* 2020) matches and averages the hidden elements. This method seems to work for a wide range of machine learning models, including CNNs and LSTMs. FedProx (Li T *et al.* 2018) adds a restriction factor between local and global weights. These methods can work better than FedAvg when facing non-IID datasets (FedProx can also reduce the effect of data or model poisoning), but all of them are still worse than the centralized implementation.

We also have thought about shifting evaluation methods to the clients. Ideally, if the testing set is provided and tested by the clients, aggregating evaluation results will be the same as performing testing in the central. But in this case, we cannot evaluate the honesty of the clients. Thus, a valid centralized testing set is still a necessity for federated learning.

We successfully implemented a federated procedure on the cloud and get similar trends as in numerical tests. However, in this case, our weight size is only 80.6 Mb and the system contains only 3 clients. There might be some limitations in transporting size, though usually, deep learning models will not be that large. Another student in the Lab is focusing on communication issues.

Also, the training design for DR detection seems to be less stable than the design of CIFAR-10 classification. Since the DR dataset is much more complex than CIFAR-10 dataset, we use a larger network with pre-trained weights and a relatively small learning rate. The pre-trained network has already provided information on basic-level features (i.e. angles, borders). Compared to CIFAR-10 classification, DR detection is more sensitive to parameter changes. Slight changes in learning rate or not using pre-trained weights can case a failer in converging. Since the size of the DR dataset is relatively small for classifying complex images, the system can be weaker in resisting poisoning or delayed nodes, but this is a hypothesis that needs further research.

7 Conclusion

We have tested several scenarios in federated learning under various circumstances, the results can provide an easy-to-understand reference to future federated designs. Our simulation system can also be edited for other scenario testings with tiny changes.

We also perform a trivial example of diabetic retinopathy detection on the cloud. Currently, we have not found many federated implementations in biological and medical fields. Our code can be an easy-to-understand example for bioinformaticians or other people who would like to attempt on federated machine learning.

8 Acknowledgment

I want to acknowledge suggestions and helps from my supervisor and subject reader and members in the lab. And Mattias Åkesson who provide the model and instructions for simulating federated learning.

9 References

Angermueller C, P?Rnamaa T, Parts L, Stegle O. 2016. Deep learning for computational biology. Molecular Systems Biology 12: 878.

Bagdasaryan E, Veit A, Hua Y, Estrin D, Shmatikov V. 2018. How To Backdoor Federated Learning. arXiv: Cryptography and Security

Benitez CMV, Chidambaram C, Hembecker F, Lopes HS. 2011. A Comparative Study of Machine Learning and Evolutionary Computation Approaches for Protein Secondary Structure Classification. InTech

Chen X, Ishwaran H. 2012. Random forests for genomic data analysis. Genomics 99:

Cubuk ED, Zoph B, Mane D, Vasudevan VK, Le QV. 2018. AutoAugment: Learning Augmentation Policies from Data. arXiv: Computer Vision and Pattern Recognition

Eraslan G, Simon LM, Mircea M, Mueller NS, Theis FJ. 2019. Single-cell RNA-seq denoising using a deep count autoencoder. Nature Communications 10: 390.

Fang M, Cao X, Jia J, Gong NZ. 2019. Local Model Poisoning Attacks to Byzantine-Robust Federated Learning. arXiv: Cryptography and Security

Gu T, Liu K, Dolangavitt B, Garg S. 2019. BadNets: Evaluating Backdooring Attacks on Deep Neural Networks. IEEE Access 7: 47230–47244.

He Chaoyang, M. Annavaram, Avestimehr S. 2020. FedNAS: Federated Deep Learning via Neural Architecture Search. CVPR 2020 workshop

Jeong E, Oh S, Kim H, Park J, Bennis M, Kim S. 2018. Communication-Efficient On-Device Machine Learning: Federated Distillation and Augmentation under Non-IID Private Data. arXiv: Learning

Jurtz VI, Johansen AR, Nielsen M, Almagro Armenteros JJ, Nielsen H, Sønderby CK, Winther O, Sønderby SK. 2017. An introduction to deep learning on biological sequence data: examples and solutions. Bioinformatics 33: 3685–3690.

Kairouz P, Mcmahan HB, Avent B, Bellet A, Bennis M, Bhagoji AN, Bonawitz K, Charles Z, Cormode G, Cummings R, others. 2019. Advances and Open Problems in Federated Learning. arXiv: Learning

Karimireddy SP, Kale S, Mohri M, Reddi SJ, Stich SU, Suresh AT. 2019. SCAFFOLD: Stochastic Controlled Averaging for On-Device Federated Learning. arXiv: Learning

Krizhevsky A. 2012. Learning Multiple Layers of Features from Tiny Images. University of Toronto

Li L, Xu W, Chen T, Giannakis GB, Ling Q. 2018. RSA: Byzantine-Robust Stochastic Aggregation Methods for Distributed Learning from Heterogeneous Datasets.

Li T, Gao Y, Wang K, Guo S, Liu H, Kang H. 2019a. Diagnostic Assessment of Deep Learning Algorithms for Diabetic Retinopathy Screening. Information Sciences 501: 511–522.

Li T, Sahu AK, Talwalkar A, Smith V. 2019b. Federated Learning: Challenges, Methods, and Future Directions. arXiv: Learning

Li T, Sahu AK, Zaheer M, Sanjabi M, Talwalkar A, Smith V. 2018. Federated Optimization for Heterogeneous Networks. arXiv: Learning

Li T, Sanjabi M, Smith V. 2019c. Fair Resource Allocation in Federated Learning. arXiv: Learning

Li Y, Han R, Bi C, Li M, Wang S, Gao X. 2018. DeepSimulator: a deep simulator for Nanopore sequencing. Bioinformatics 34: 2899–2908.

Liu Y, Chen T, Yang Q. 2018. Secure Federated Transfer Learning. arXiv: Learning

Mcmahan HB, Moore E, Ramage D, Hampson S, Arcas BAY. 2016. Communication-Efficient Learning of Deep Networks from Decentralized Data. arXiv: Learning

Nishio T, Yonetani R. 2019. Client Selection for Federated Learning with Heterogeneous Resources in Mobile Edge. 1–7.

Passeratpalmbach J, Farnan T, Miller RC, Gross MS, Flannery HL, Gleim B. 2019. A blockchain-orchestrated Federated Learning architecture for healthcare consortia. arXiv: Computers and Society

Peng X, Huang Z, Zhu Y, Saenko K. 2019. Federated Adversarial Domain Adaptation. arXiv: Computer Vision and Pattern Recognition

Poplin R, Chang P-C, Alexander D, Schwartz S, Colthurst T, Ku A, Newburger D, Dijamco J, Nguyen N, Afshar PT, Gross SS, Dorfman L, McLean CY, DePristo MA. 2018. A universal SNP and small-indel variant caller using deep neural networks. Nature Biotechnology 36: 983–987.

Rajpurkar P, Irvin J, Ball RL, Zhu K, Yang B, Mehta H, Duan T, Ding D, Bagul A, Langlotz CP, Patel BN, Yeom KW, Shpanskaya K, Blankenberg FG, Seekins J, Amrhein TJ, Mong DA, Halabi SS, Zucker EJ, Ng AY, Lungren MP. 2018. Deep learning for chest radiograph

diagnosis: A retrospective comparison of the CheXNeXt algorithm to practicing radiologists. PLOS Medicine 15: 1–17.

Ramachandran P, Zoph B, Le QV. 2017. Swish: a Self-Gated Activation Function. arXiv: Neural and Evolutionary Computing

Real E, Liang C, So D, Le QV. 2020. AutoML-Zero: Evolving Machine Learning Algorithms From Scratch. arXiv: Learning

Sheller MJ, Reina GA, Edwards B, Martin J, Bakas S. 2018. Multi-Institutional Deep Learning Modeling Without Sharing Patient Data: A Feasibility Study on Brain Tumor Segmentation. arXiv: Learning

Stokes JM, Yang K, Swanson K, Jin W, Cubillos-Ruiz A, Donghia NM, MacNair CR, French S, Carfrae LA, Bloom-Ackermann Z, Tran VM, Chiappino-Pepe A, Badran AH, Andrews IW, Chory EJ, Church GM, Brown ED, Jaakkola TS, Barzilay R, Collins JJ. 2020. A Deep Learning Approach to Antibiotic Discovery. Cell 180: 688-702.e13.

Suresh AT, Mcmahan B, Kairouz P, Sun Z. 2019. Can You Really Backdoor Federated Learning. arXiv: Learning

Wang H, Yurochkin M, Sun Y, Papailiopoulos DS, Khazaeni Y. 2020. Federated Learning with Matched Averaging.

Wu Z, Ling Q, Chen T, Giannakis GB. 2019. Federated Variance-Reduced Stochastic Gradient Descent with Robustness to Byzantine Attacks. arXiv: Learning

Xie C, Huang K, Chen P, Li B. 2020. DBA: Distributed Backdoor Attacks against Federated Learning.

Xu M, Zhao Y, Bian K, Huang G, Mei Q, Liu X. 2020. Federated Neural Architecture Search. arXiv

Yao Q, Wang M, Escalante HJ, Guyon I, Hu Y, Li Y, Tu W, Yang Q, Yu Y. 2018. Taking Human out of Learning Applications: A Survey on Automated Machine Learning. arXiv: Artificial Intelligence

Yurochkin M, Agarwal M, Ghosh S, Greenewald K, Hoang TN, Khazaeni Y. 2019. Bayesian Nonparametric Federated Learning of Neural Networks. arXiv: Machine Learning

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

10 Example Distribution

2 class non-IID, 4k dataset (CIFAR-10), 10 worker

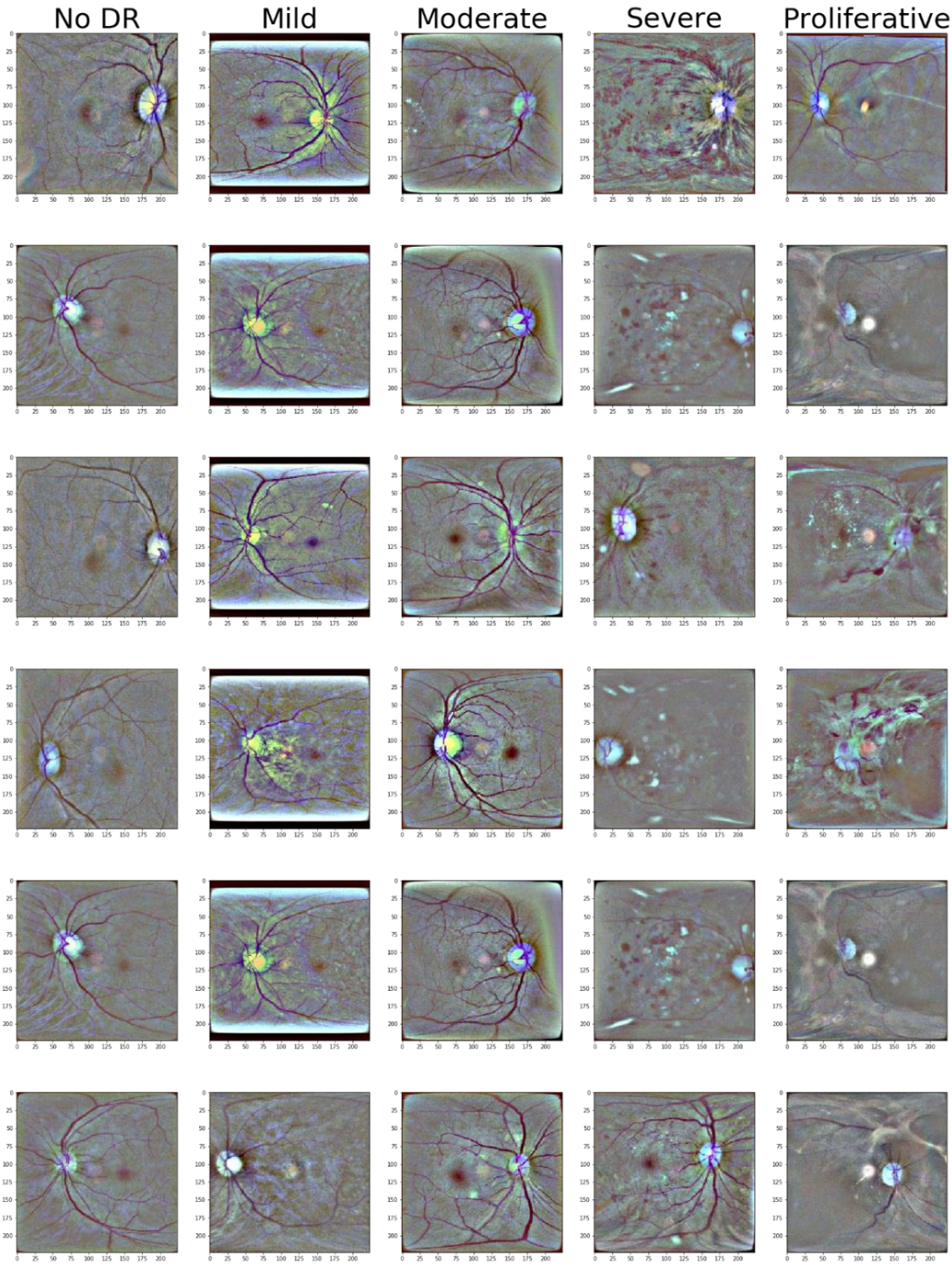
Node 1	Classe 1 * 200 + Class 2 *200
Node 2	Classe 2 * 200 + Class 3 *200
Node 3	Classe 3 * 200 + Class 4 *200
Node 4	Classe 4 * 200 + Class 5 *200
Node 5	Classe 5 * 200 + Class 6 *200
Node 6	Classe 6 * 200 + Class 7 *200
Node 7	Classe 7 * 200 + Class 8 *200
Node 8	Classe 8 * 200 + Class 9 *200
Node 9	Classe 9 * 200 + Class 10 *200
Node 10	Classe 1 * 200 + Class 10 *200

*No overlap in samples between nodes.

*Each client is a node.

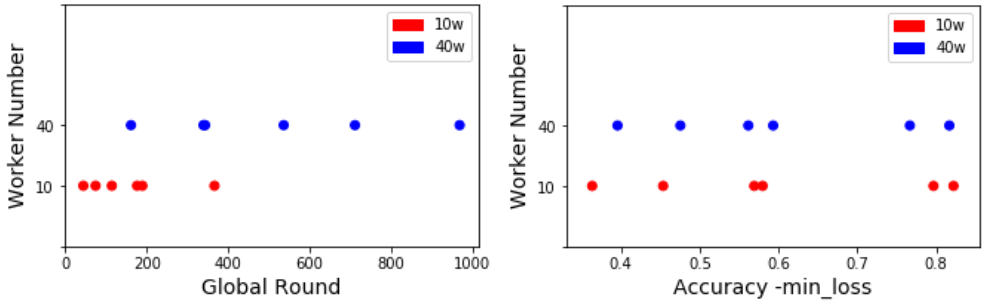
11 Appendix - OIA-DDR dataset after pre-processing

Methods from (https://www.kaggle.com/ratthachat/aptos-eye-preprocessing-in-diabetic-retinopathy) and (https://www.kaggle.com/titericz/circle-to-rectagle-preprocessing-1) are used for preprocessing original images to our input data.

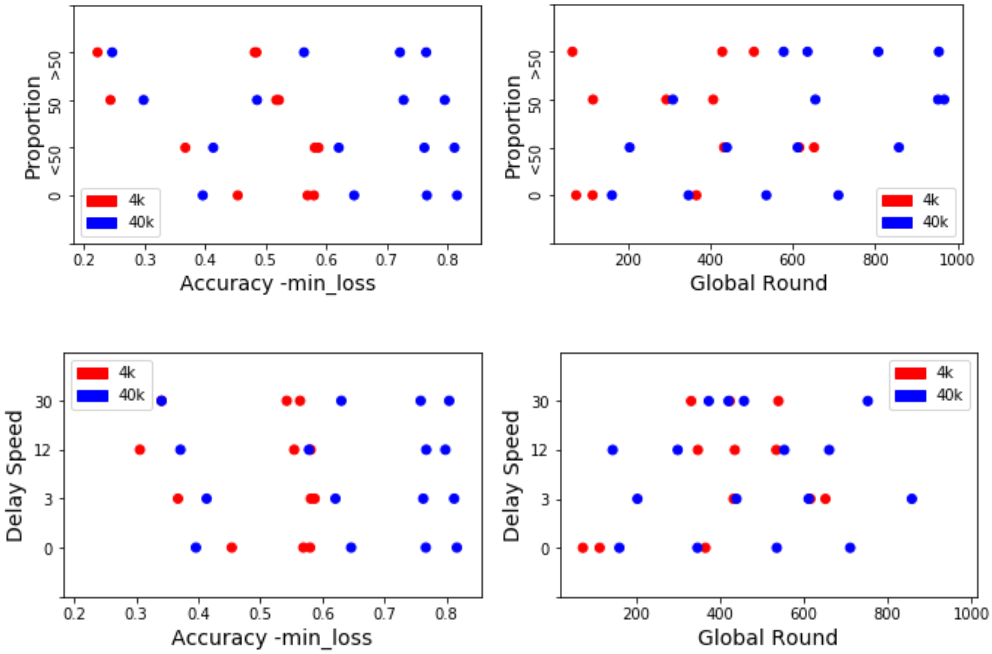


12 Appendix – Samples used in Box plots

Client Size



Delayed Update



13 Appendix - Statistical Records

In this Appendix, we record the results of statistical tests. Shapiro–Wilk test and Levene's test have been performed to assess normality and variance homogeneity, which is the pre-requirement of using the T-test. We perform the T-test or Wilcoxon test depending on assessing results. Results tested by the Wilcoxon test are tagged with '#'.

Client Size

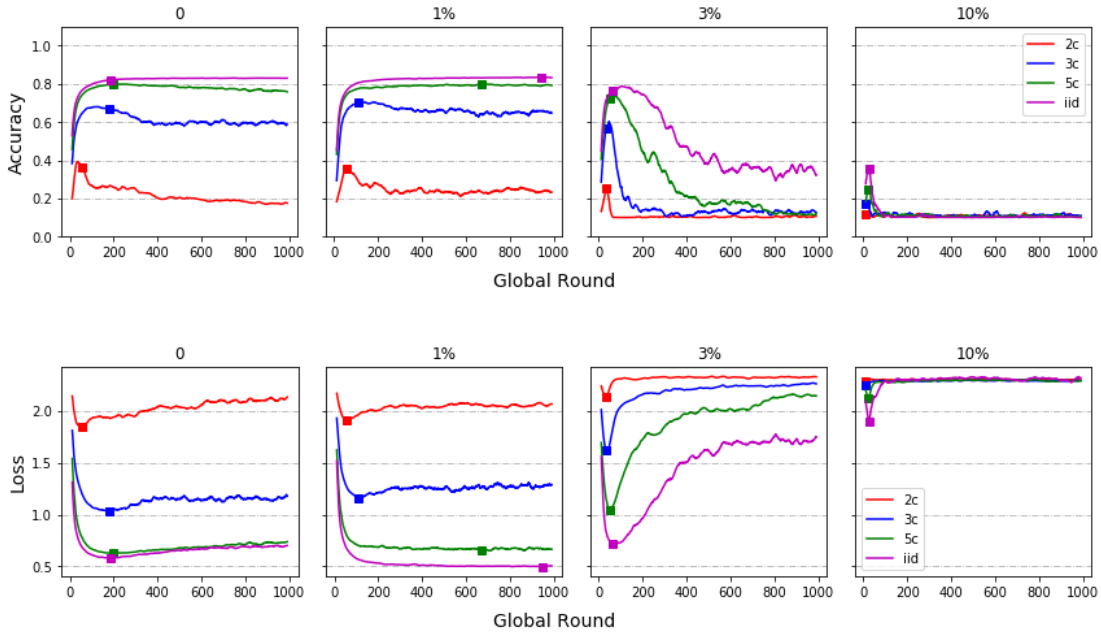
	'10W'-'40W'
ACCURACY	0.71
ROUND	0.0059 *

Delayed Update

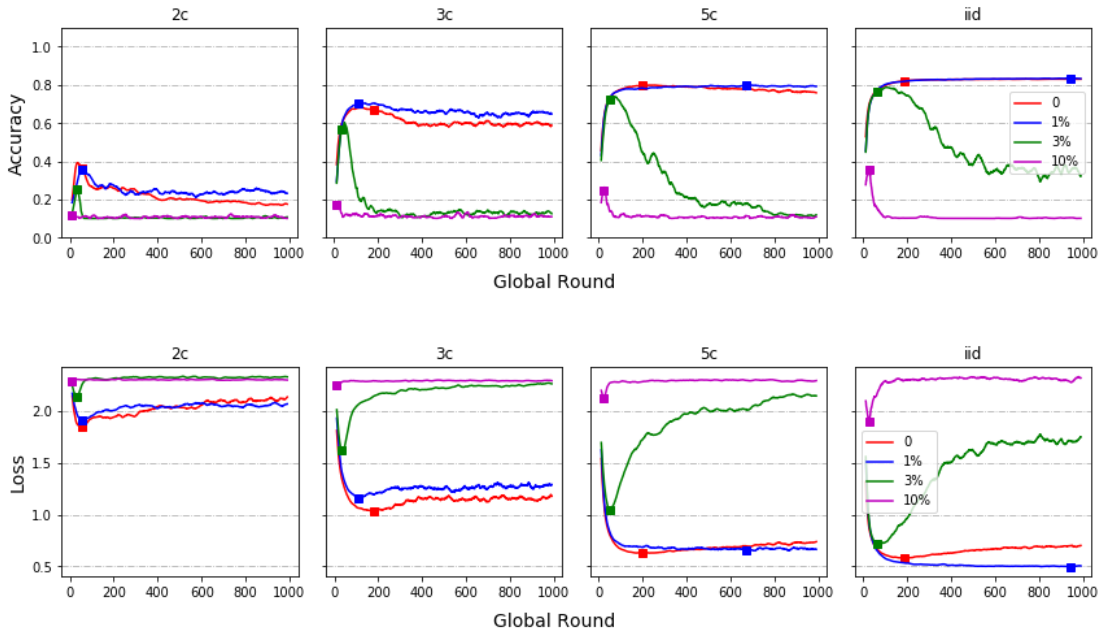
PROPORTION	**'0'-'<50'**	**'<50'-'50'**	**'50'-'>50'**	**'0'-'50'**	**'0'-'>50'**	**ANOVA**
ACCURACY	0.41	0.0041 *	0.39	0.01 *	0.005 *	0.62
ROUND	0.016 *	0.90	0.60	0.089	0.092	0.35
DELAY SPEED	**'0'-'3'**	**'3'-'12'**	**'12'-'30'**	**'0'-'12'**	**'0'-'30'**	**ANOVA**
ACCURACY	0.41	0.027 *	0.77	0.11	0.052	0.97
ROUND	0.016 *	0.014 *	0.44	0.17	0.059	0.23

14 Appendix - Scenario testing results with 40k set

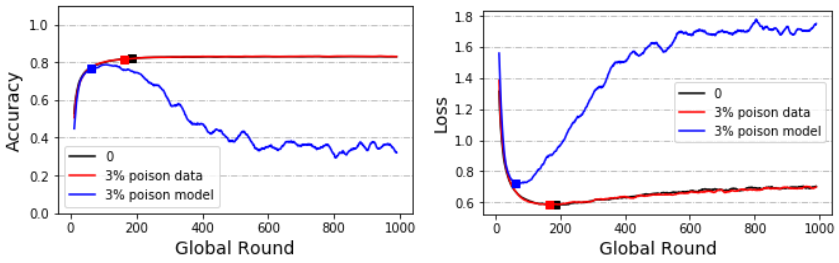
Model Poisoning – By Anomaly Node Size



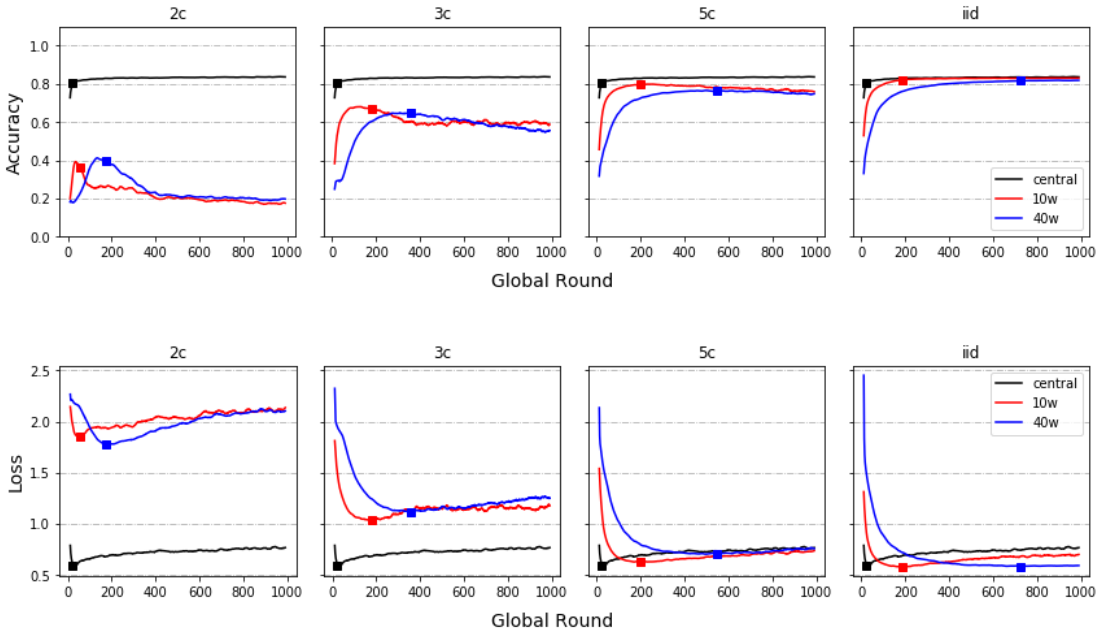
Model Poisoning – By IID/non-IID



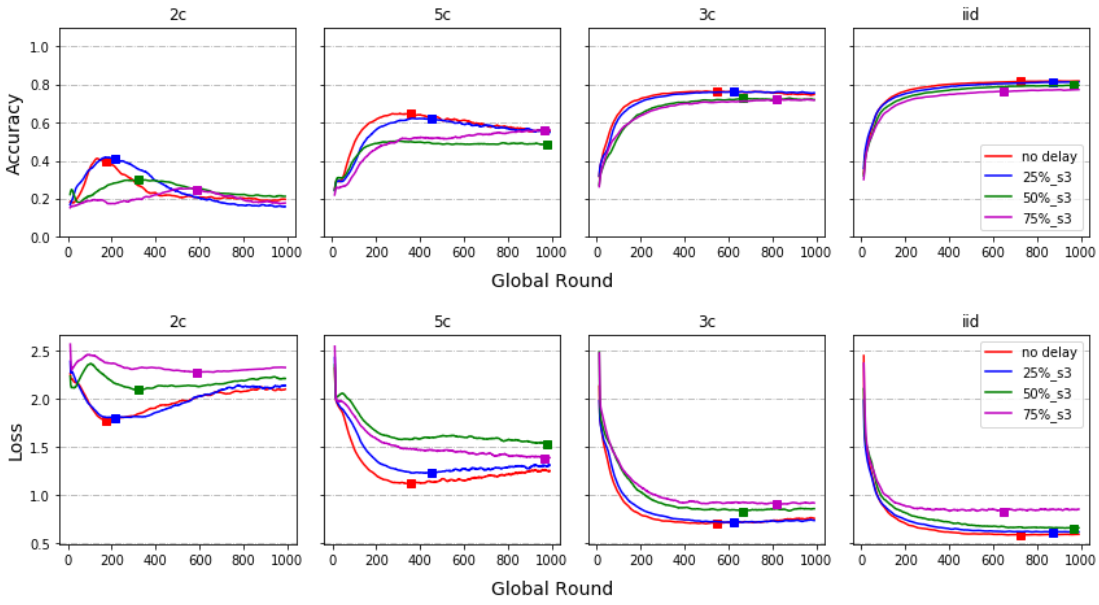
Data Poisoning



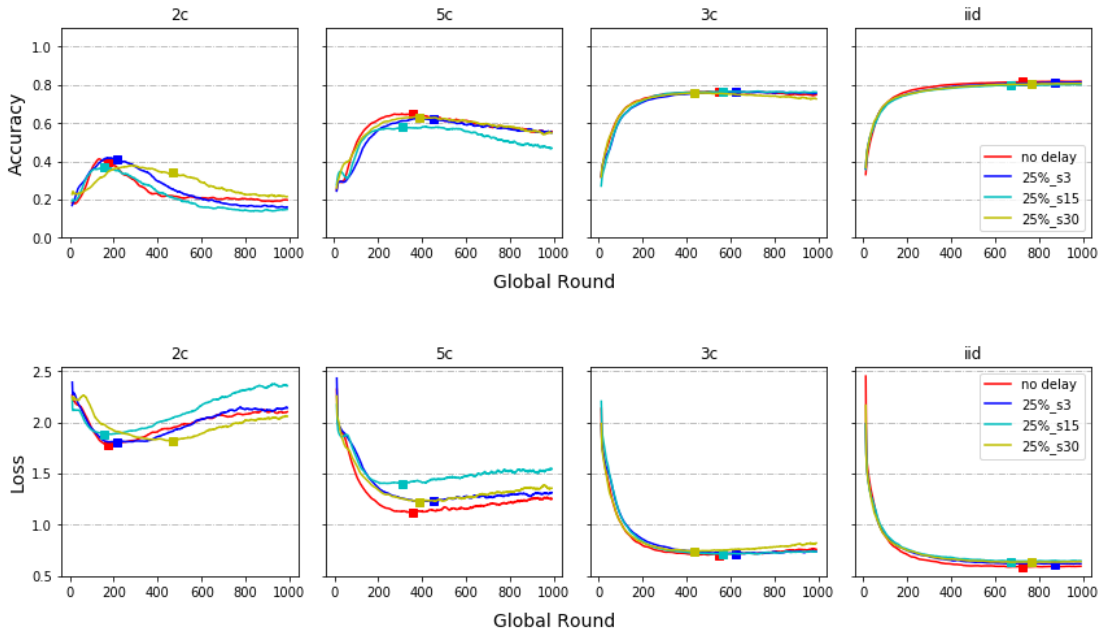
Client Size



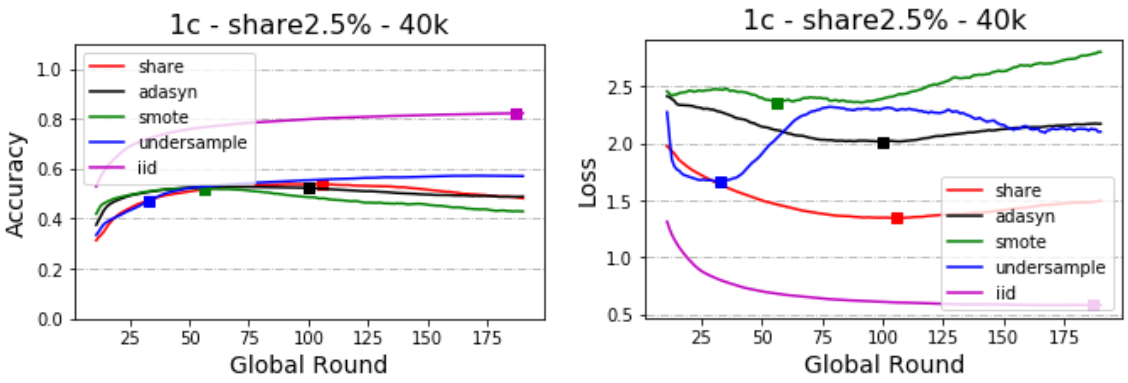
Delayed Update – By Proportion



Delayed Update – By Speed

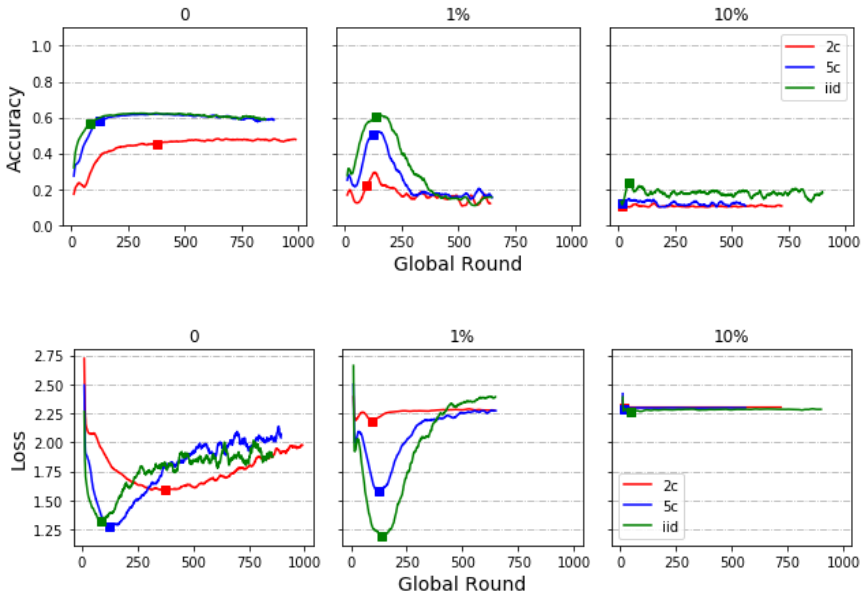


Share data strategy (Additional)

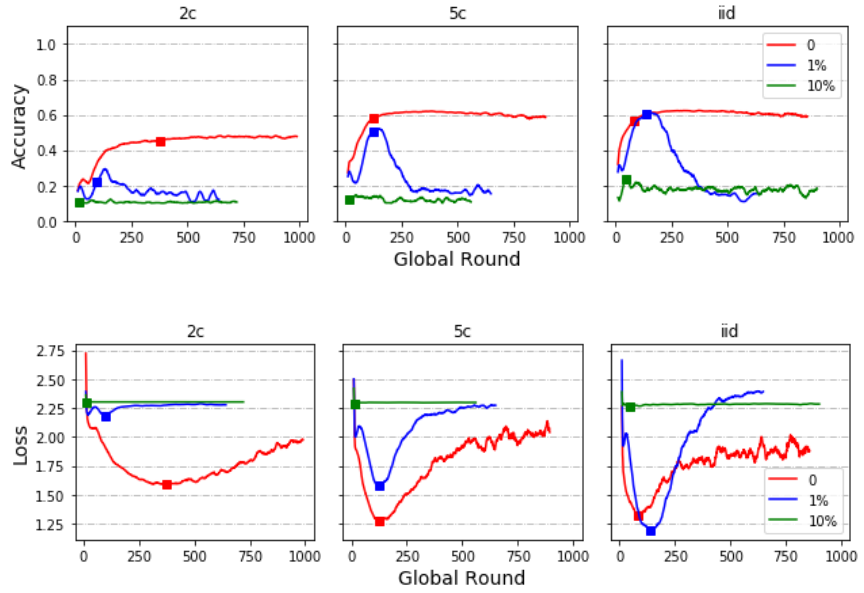


15 Appendix - Scenario testing results with 4k set

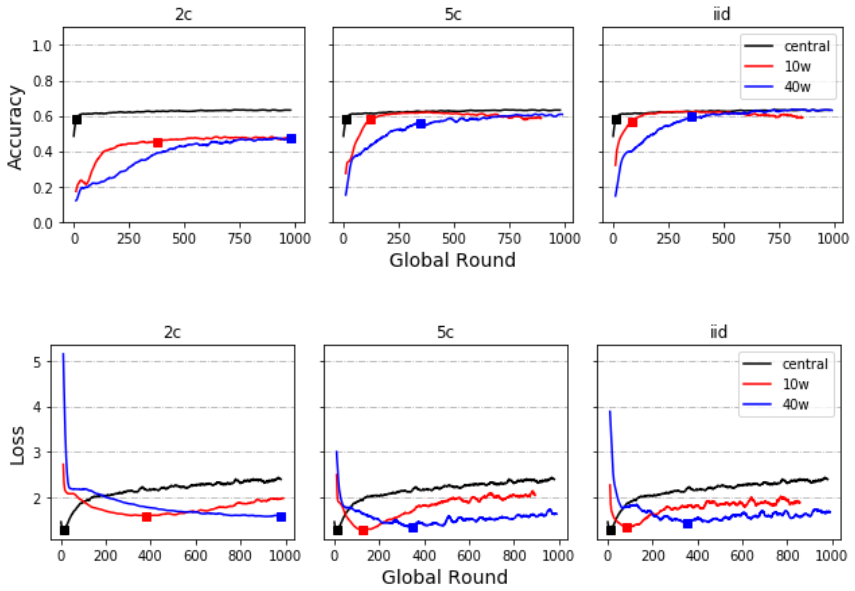
Model Poisoning – By Anomaly Node Size



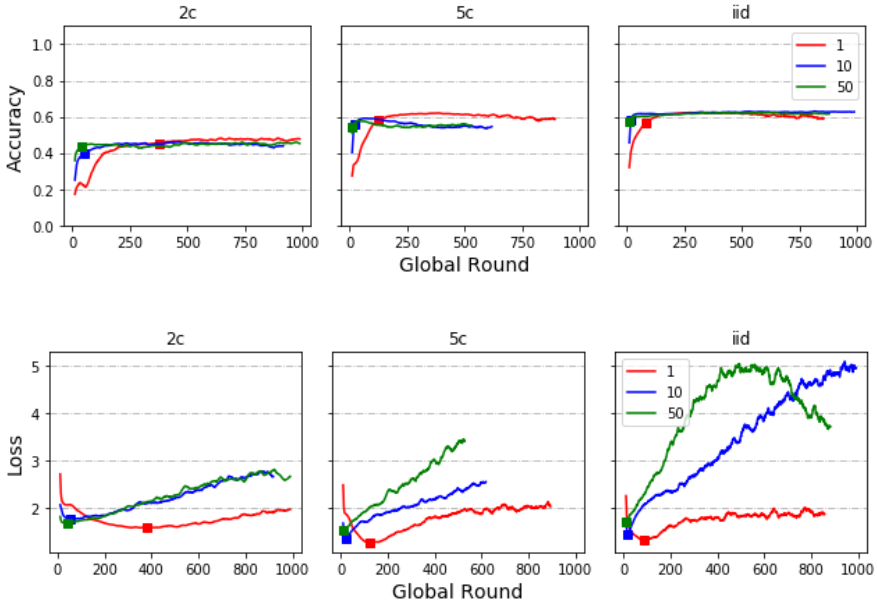
Model Poisoning – By IID/non-IID



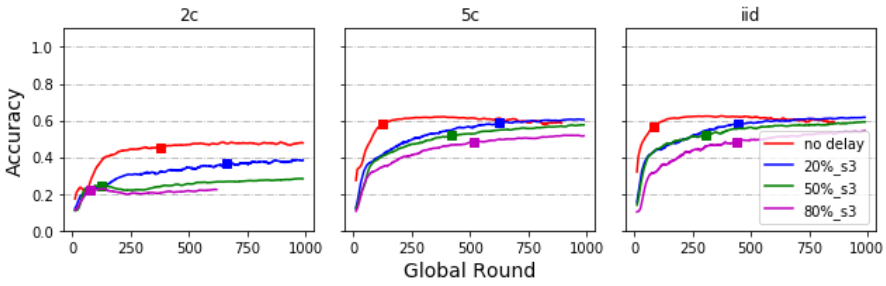
Client Size

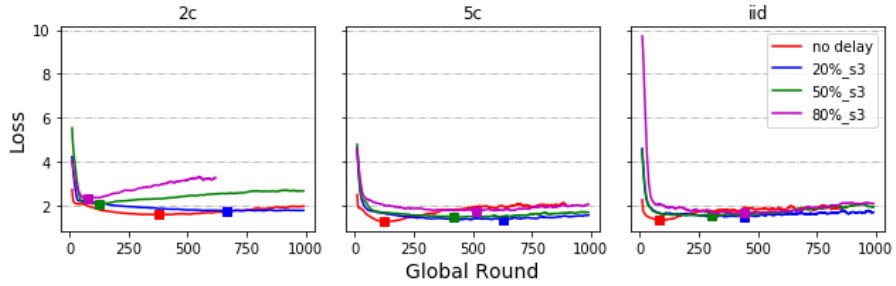


Local Round (Additional)

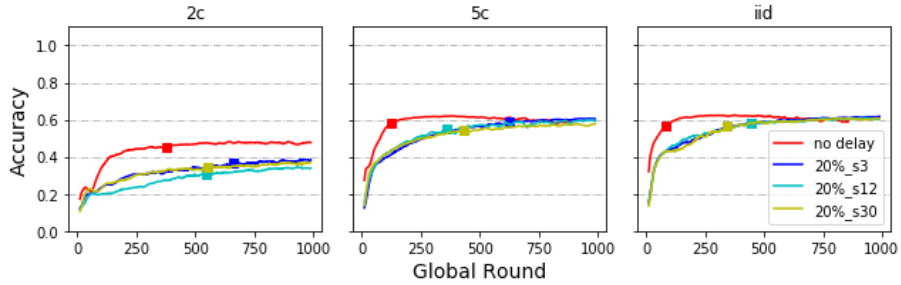


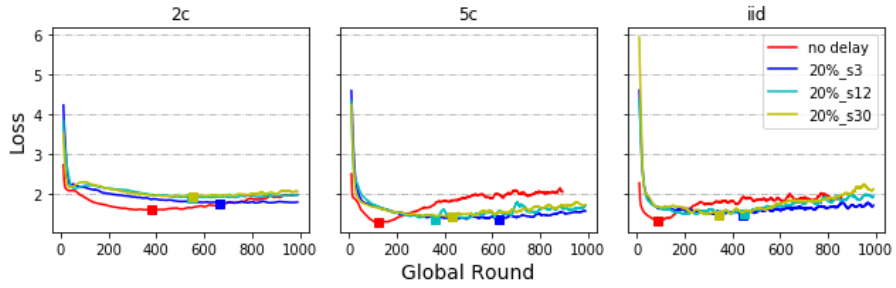
Delayed Update – By Proportion



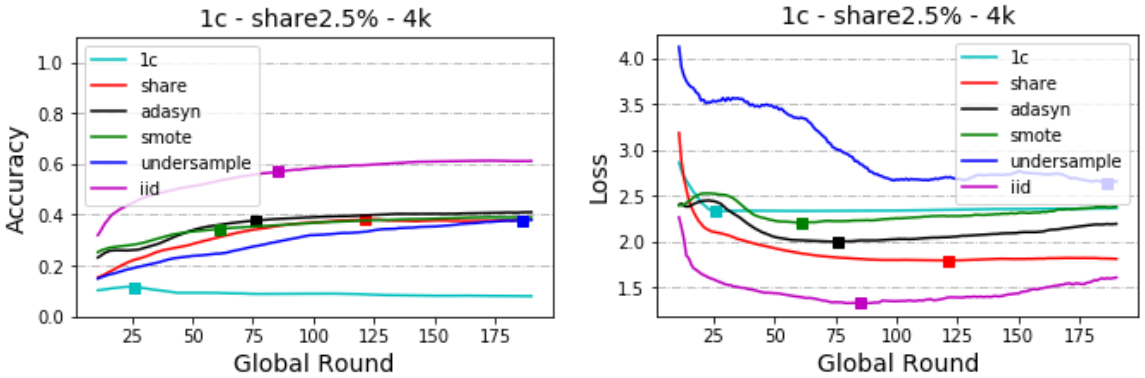


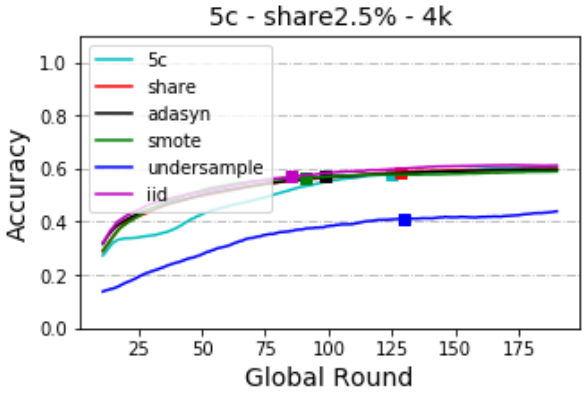
Delayed Update – By Speed

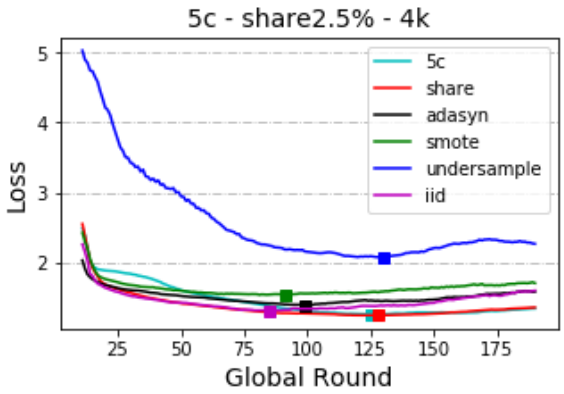




Share data strategy




5c - share2.5% - 4k
Accuracy
Global Round
5c
share
adasyn
smote
undersample
iid
0.0
0.2
0.4
0.6
0.8
1.0
25
50
75
100
125
150
175


5c - share2.5% - 4k
Loss
Global Round
5c
share
adasyn
smote
undersample
iid
2
3
4
5
25
50
75
100
125
150
175

16 Appendix - Results for cloud implementation