1. Introduction

Machine learning methods has been applied to medical or biological researches for long. Traditional algorithms such as PCA, random forest, AdaBoost has been widely used in bioinformatic researches (Benitez *et al.* 2011, Chen & Ishwaran 2012). In recent years, a sub field called deep learning has become 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. Beside these, many interesting works has been done in recent years. DeepVariant aims to detect genetic variants from sequencing data (Poplin *et al.* 2018). DeepSimulator attempts to generate 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).

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 over an empirical dataset of size 2335 (Stokes *et al.* 2020). In many other cases, 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 access the appropriate dataset. In many cases, datasets are small and unbalanced. In other cases, annotations are not suitable for the specific training task. Many institutes 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 will test federated learning’s behavior under different scenarios and perform an example federated medical image classification task on cloud. We hope this work can provide refences for future federated designs and provide a trivial example for building up a federated system.

There are several challenges in federated learning, including communication costs, security

protection, statistical and systems heterogeneity (Li T *et al.* 2019b). In this project, we will not put too much attention on communication or security issues.

1. Background

2.1 Federated Learning

Federated learning is a machine learning strategy where many clients 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 central server and the most importantly, preserves each client’s private data.

There are many domains of federated learning. Cross-device federated learning is implemented on large number of mobile or IoT 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 shared. 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 (Yang *et al.* 2019).

In most biological or medical use cases, cross-silo and horizontal federated learning will be performed. A success case of brain tumor segmentation in federated settings has been reported (Sheller *et al.* 2018).

2.2 Federated Algorithms

Federated Averaging (FedAvg) and Federated SGD (FedSGD) are basics for many optimization algorithms and are widely used. In FedAvg settings, client weights will be collected and averaged while in FedSGD settings, gradients will be collected. Many other optimization algorithms are developed for various purposes, (Kairouz *et al.* 2019) provides a summary of popular ones.

2.3 Federated Frameworks

In the appendix of (Kairouz *et al.* 2019), several popular federated frameworks for both simulation and production are summarized. TensorFlow Federated and PySyft provide tools for federated learning based on the TensorFlow and PyTorch software. 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. FATE already have enabled several functions for both traditional learning and deep learning.

2.4 Communication tools

In federated system, clients need to communicate with the server. TensorFlow Federated uses gRPC and 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 will use gRPC for communication.

1. Datasets and Methods

In this project, the CIFAR-10 dataset (Krizhevsky 2012) has been used for locally simulating different scenarios of federated learning, the OIA-DDR dataset (Li T *et al.* 2019a) has been used for implementing an example medical image classification task on the cloud. The performance of the models will not be high enough for any actual usage, only for testing federated learning in this project.

Each simulation task has been tested for 1000 epochs (for centralized training) or global rounds (for federated learning) and the cloud implementation has been tested for 200 epochs/rounds. The code is written in Python3 and TensorFlow 2.0.1 is used.

3.1 CIFAR-10 dataset

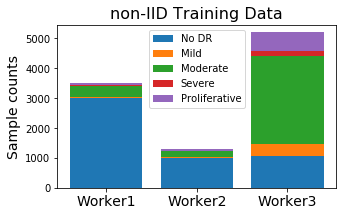
The CIFAR-10 dataset consists of 50,000 training images and 10,000 testing images of 10 balanced classes, each image is in color format and the size is 32x32 pixels.

3.2 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 public kernels on Kaggle, crop and resize them to 224x224 pixels. (<https://www.kaggle.com/ratthachat/aptos-eye-preprocessing-in-diabetic-retinopathy>) (<https://www.kaggle.com/titericz/circle-to-rectagle-preprocessing-1>)

In our implementation, we select 10000 balanced samples and divide it to non-IID federate training set. The rest 2521 samples are used for testing.

(Figure xxx each class’s count)

3.3 Training designs

The task for CIFAR-10 is to input images of size 32x32x3 and classify them to 10 classes. The model is provided by Mattias Åkesson. We apply ADAM optimizer with learning rate set to 1e-3. The batch size is set between 50 to 100; loss will be calculated by categorical cross-entropy.

The task for OIA-DDR dataset is to input images of size 224x224x3 and classify them to 2 classes. VGG16 network (pre-trained on ImageNet dataset) is used. We use ADAM optimizer with learning rate initialized at 1e-4 and decayed by rate 0.05 in every epoch or round. The batch size is set to 300; loss will be calculated by binary cross-entropy.

learning\_rate = 1e-4/(1+0.05\*epoch\_number)

3.4 Scenarios testing

We simulate and test different scenarios in federated learning in this part. Due to the limit of time and computational resources, each test is performed only once.

***Simulation System***

One central node and several client nodes are involved in this simulation system. We assume no data or nodes will be dropped out from or added into the system during the entire training process.

When the system starts, the central node will initialize a model. We apply Federated Averaging in this system, weighted by node’s size. Each global round consists of the following basic actions:

1. Client nodes load weight from the central node,
2. Client nodes train the model over their local data and save the history,
3. Client nodes return weights to the central nodes,
4. Central node aggregates the weights and evaluates the model.

After simulation, we collect the evaluation results in central node’s folder as the overall testing history and average the node’s training history as the overall training history.

***Pilot Tests***

Before the formal implementation of different scenarios, we first test the system with different sizes of dataset and different non-IID classes. Figure xxx compares centralized and federated learning’s behavior with different size of training dataset. The result fluctuates due to randomness, but it is obvious that the performance of both federated and centralized learning can be improved with larger training set, and federated learning always converge at slower speed than the centralized learning.

Figure xxx

The figure shows the training result with different size of dataset. (window size=21) LEFT: This plot shows the windowed accuracy at minimum windowed loss. RIGHT: This plot shows the global rounds to reach to minimum windowed loss.

In previous works we already know that a non-IID training set can result in performance degradation in federated learning. There is no obvious standard to quantify the degree of non-IID, but researchers usually identify it by available classes in each node. Figure xxx shows federated learning’s behavior at different non-IID levels. It is obvious that accuracy and speed will decrease as the degree of non-IID increases.

(Figure xxx. Pilot test)

LEFT: This figure shows the windowed accuracy of the global round with minimum windowed loss for different non-IID classes of the first 500 rounds. RIGHT: This figure shows the global rounds to reach to minimum windowed loss for different non-IID classes of the first 500 rounds (window size=21).

We also have observed different behaviors between tests using dataset of size 4000 and 40000 under 2 class non-IID circumstances. In other circumstances, training with larger dataset always result in higher accuracy.



Figure xxx

4k v.s. 40k dataset under 2 classes non-IID condition.

***Default Simulation Settings***

We will select a balanced total training set of size 4000 or 40000 and apply evaluation with Cifar10’s whole testing set. Training set are distributed to 10 or 40 nodes depending on the experimental design. Client nodes are in equal size and samples from available classes within a node are balanced. By default, local epochs are set to 1 and global rounds are set to 1000. Evaluation is performed by the central node. In each scenario, we change 1 parameter and compare it to the one without that change.

Scenarios are tested under 2 classes, 5 classes and IID classes circumstances with dataset of size 4000 (4k dataset) and under 2 classes, 3 classes, 5 classes and IID classes circumstances with dataset of size 40000 (40k dataset). Early stopping of 500 rounds is applied to testing with 4k dataset.

***Model Poisoning***

We test this scenario where there is an abnormal node that returns arbitrary weights at each global round. We split the training set to 10 nodes and added in an abnormal node who claims to be 1% or 10% as large as total training dataset.

Testing is performed with 4k and 40k dataset. Additional tests of 3% abnormal node size are performed with 40k dataset. Other addition tests of poisoning frequency and poisoning data are performed with 40k dataset under IID condition.

***Data Dispersion***

In some tests we distribute our data to 10 clients, in other tests to 40 clients. In this scenario, we compare the training performance between 10 client and 40 client cases in default settings.

***Delayed Update***

In this scenario, some nodes update at slower speed than others. We split the training set to 40 nodes and test the situation from 2 aspects:

1. Different proportion of delayed nodes: 25%, 50% or 75% nodes delayed, they load the weights at global round n, and upload the weights at global round n+3. New weights won’t be loaded until current weights are uploaded.
2. Different speed of delayed nodes: 20% nodes delayed, they load the weights at global round n, and upload the weights at global round n+3, n+12 or n+30. New weights won’t be loaded until current weights are uploaded.

***Share data strategy***

In previous tests, 1 class non-IID never behaves better than random. A previous research suggested a data-sharing strategy that can improve the training performance over 1 class non-IID data (Zhao *et al.* 2018). The author assumes a small IID set (2.5% to 25% as large as total training dataset) can be published for pre-training the model or sharing between nodes.

In this scenario, we attempt to different methods of using the shared data. 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 resample an IID set from the pool. Method 3 will duplicate small classes in the pool to make the pool label balanced. Test is performed under 1 class non-IID circumstances with 40k dataset and 10 client nodes, dataset to be shared is as large as 2.5% of the total set.

Tests with other sizes of shared set, other non-IID or IID circumstances or testing with the 4k dataset is also performed.

3.5 Cloud implementation

In the cloud implementation experiment, our synchronous gRPC system consists of 1 central server and 3 clients. The clients join in the training by sending request to the server at every round. The server fixes its client number to 3, aggregation will start when 3 returned weights are received. Each global round consists of the following basic actions (from the server’s side):

1. The server initializes the model and save the weight,
2. Server receive requests and ,
3. Server register clients and send weights,
4. (Clients perform local training),
5. Server receive returned weights, wait until 3 weights is ready,
6. Server aggregates weights and perform evaluation,
7. Loop for 1-5.

As we mentioned before, a slightly non-IID dataset is prepared for this task. We will also test the centralized training performance with total training test and the largest federated subset.

1. Results

4.1 Scenarios testing

In this section we will show statistical summaries or part of our testing results, the full results are available in the appendix. In an ideal training, maximum accuracy and minimal loss will be achieved at converge epoch. In our trainings, we have observed slight increase in accuracy when the loss reaches to its least, in which case the model already overfits the training data (training accuracy reaches to 1). Thus, we assume the model converge at the global round with minimal loss, the accuracy at that round represents model’s best performance.

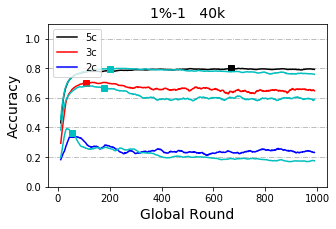
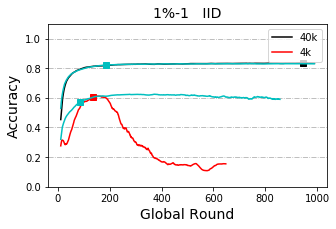
Since the original results are fluctuated, we will show and manipulate on windowed testing accuracy and loss in this report, with window size equals to 21. The full and original records and are available in the GitHub repository.

***Model Poisoning***

The presence of abnormal node leads to performance degradation. The influence is related to both the size of the abnormal node and the poisoning frequency. We label the situation when an abnormal node is as large as x% of the total training set and upload arbitrary weight by every y rounds as ‘x%-y’. We observe similar accuracy trends among situation ‘1%-1’, ‘3%-3’ and ‘10%-10’, while there is a huge difference among ‘1%-1’ , ‘3%-1’ and ‘10%-1’. In the 3 similar situations, they have the same average abnormal node size per round. The influence of abnormal node size in this case still exists. Loss of ‘10%-10’ stop decreasing at earlier rounds than ’1%-1’.



Large dataset can increase model’s tolerance to attacks from abnormal nodes. In the case of 1% size abnormal 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 difference in decrease patterns among non-IID groups has been observed.

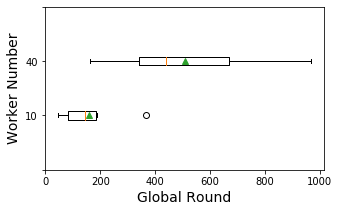
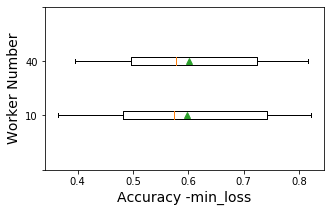
 

(light blue line is its original fed plot without any affection)

Testing results in data poisoning indicates that data poisoning is weaker than model poisoning.

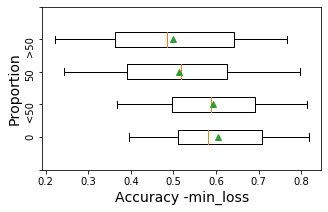


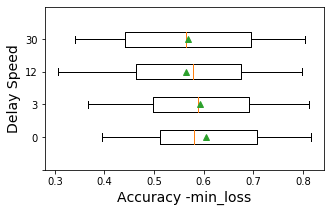
***Data Dispersion***

From Figure xxx we can see training with 40 client nodes is obviously slower than training with 10 client nodes while the convergence accuracy is not obviously affected. Paired T-test is applied and the result shows a significance in speed (p-value=0.0059). Differences in accuracy at convergence is not significant (p-value=0.71).

***Delayed Update***



Statistical tests are applied to show the difference between no-delay-node case and the other situations. T-test shows a significance between with or without delayed update in convergence speed when delay speed is 3 and proportion between 20% to 25% (p-value=0.016). Surprisingly, this significance disappears as delay speed and proportion increases. On the other hand, when grouped by delay proportion, there is also no significance between group ‘<50’ and ‘50’ or ‘50’ and ‘>50’.

When grouped by delay speed, a significance between group ‘3’ and ‘12’ has been detected. Effect on converge speed from delayed nodes is very likely to be weakened when delay speed is large.

Also, convergence accuracy with delayed proportion larger than 50 is reported to be significantly reduced (p-value=0.01 and 0.005). Delay speed does not obviously affect convergence accuracy.

***Share data***

In this scenario we compare different methods of using the shared data. As we mentioned before, each client owns a mixed pool of local and shared data. When the shared is small, data in the pool is still unbalanced.

Figure xxx shows the results between directly use the mixed dataset and do resample on the mixed dataset when sharing data is small (as large as 2.5% of total dataset). The direct method performs better than the resample method in most cases except for in 1 class non-IID scenario. In such scenario, accuracy of direct method decreases sharply after overfitting while accuracy of the resample method can stay stable.

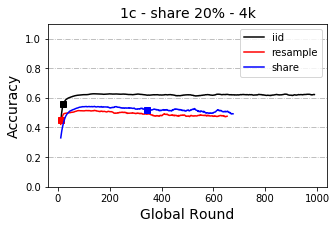
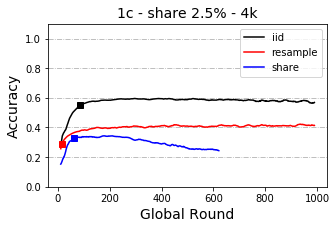
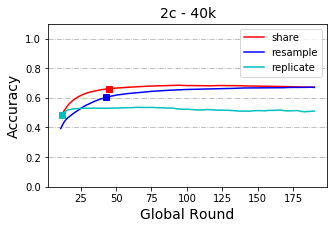
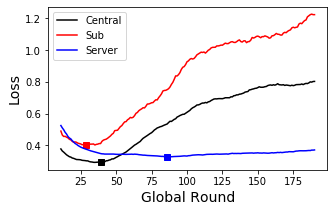
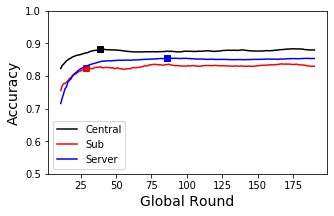


Figure xxx shows the result of replicating the small classes in the pool to make labels balanced. This method performs worst.



4.2 Cloud implementation



1. Discussion

(like how to evaluate on node.)

1. Conclusion
2. Acknowledgement
3. Reference

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

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:

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.

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

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

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

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.

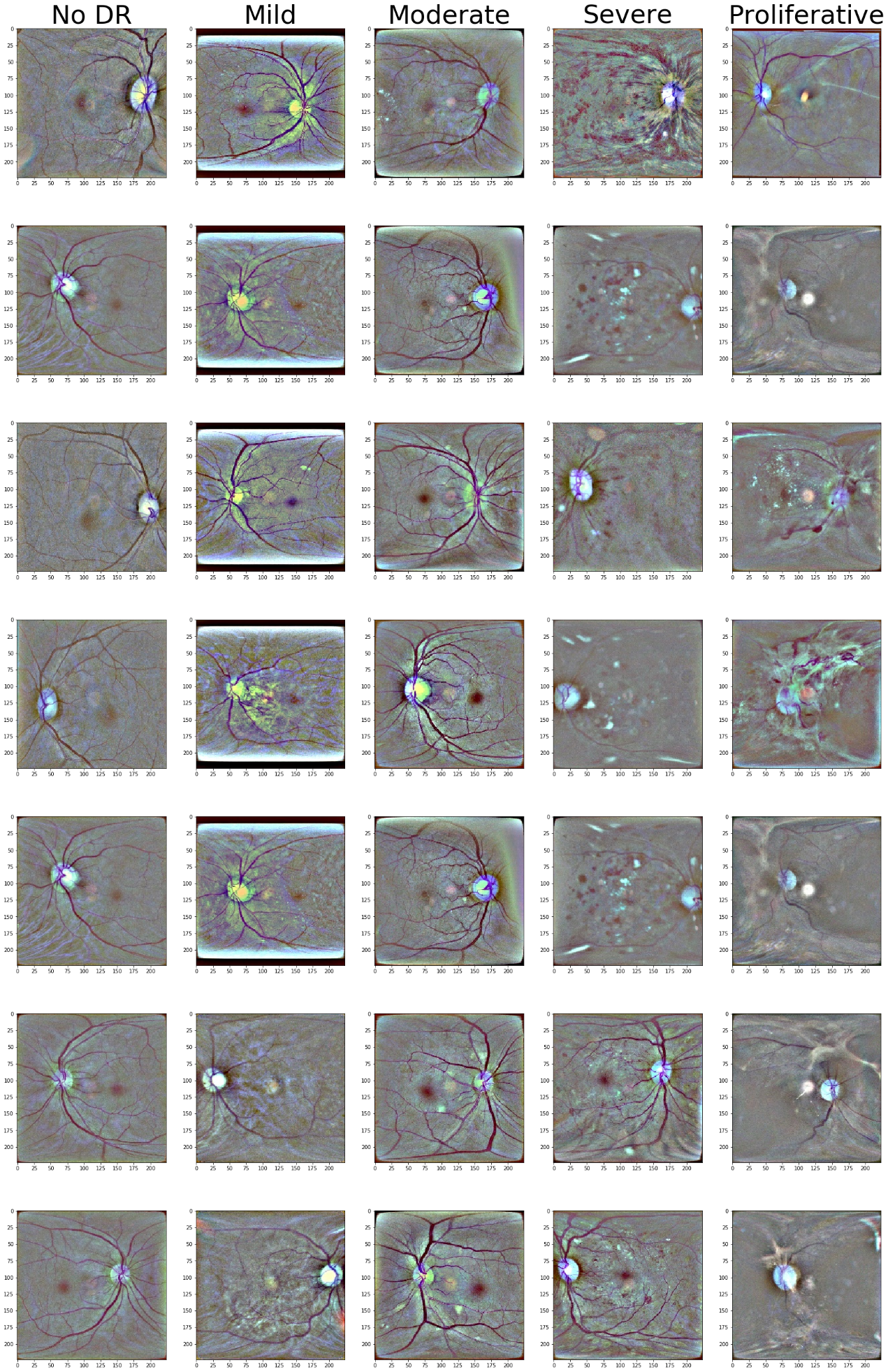
Sheller MJ, Reina GA, Edwards B, Martin J, Bakas S. 2018. Multi-Institutional Deep Learning Modeling Without Sharing Patient Data: A Feasibility Study on Brain Tumor Segmentation. arXiv: Learning

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.

Yang Q, Liu Y, Chen T, Tong Y. 2019. Federated Machine Learning: Concept and Applications. arXiv: Artificial Intelligence

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

Appendix 1 OIA-DDR dataset after preprocessing



Appendix – Dot plots

***Data Dispersion***



***Delayed Update***





Appendix –p-value 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 T-test. We perform T-test or Wilcoxon test depending on assess results. Results tested by Wilcoxon test are tagged with ‘#’.

***Data Dispersion***

|  |  |
| --- | --- |
|  | ‘10w’-‘40w’ |
| ACC | 0.71 |
| Round | 0.0059 \* |

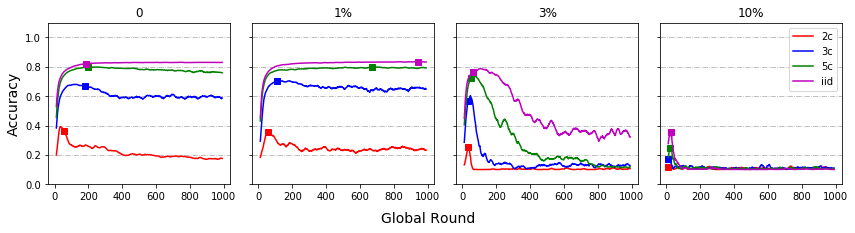
***Delayed Update***

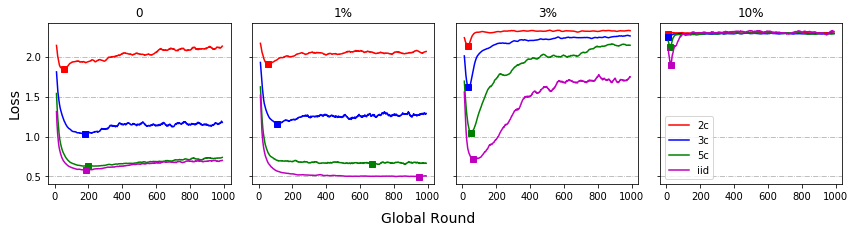
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Proportion | ‘0’-‘<50’ | ‘<50’-‘50’ | ‘50’-‘>50’ | ‘0’-‘50’ | ‘0’-‘>50’ | ANOVA |
| Acc | 0.41 | 0.0041 \* | 0.39 | 0.01 \* | 0.005 \* | 0.62 |
| Round | 0.016 \* | 0.90 | 0.60 | 0.089 | 0.092 | 0.35 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 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 |

Appendix- Scenario testing results with 40k training set

***Model Poisoning – By Abnormal Node Size***





***Model Poisoning – By IID/non-IID***



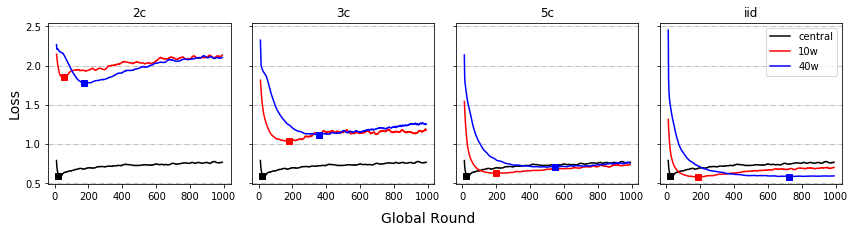


***Data Poisoning***



***Data Dispersion***





***Delayed Update – By Proportion***

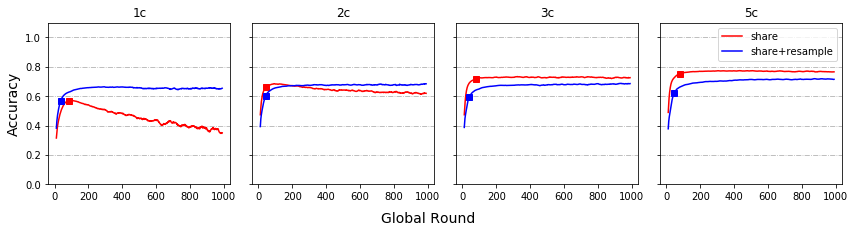


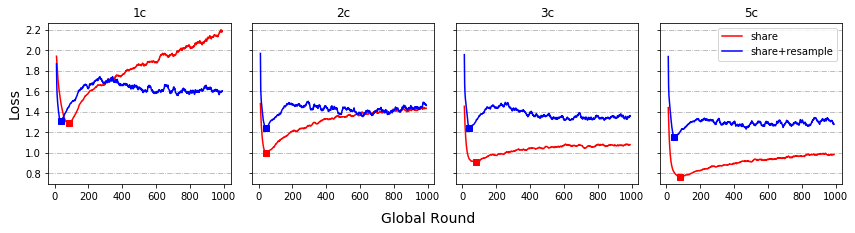
***Delayed Update – By Speed***

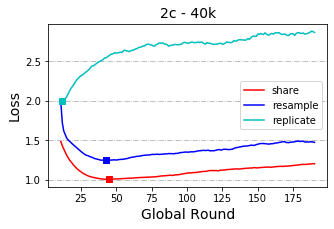
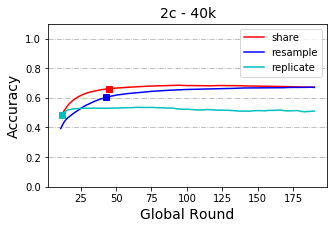




***Share data***

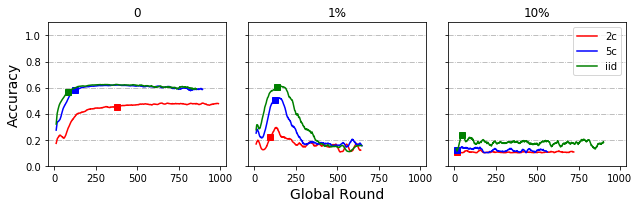


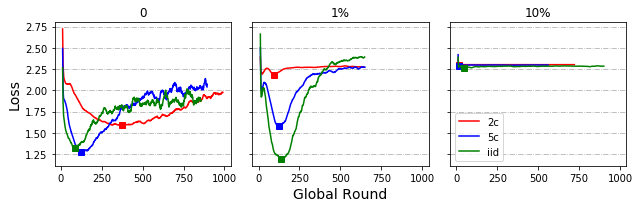




Appendix- Scenario testing results with 4k training set

***Model Poisoning – By Abnormal Node Size***





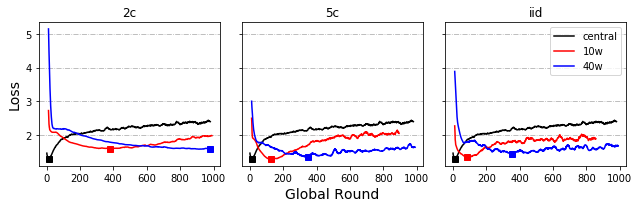
***Model Poisoning – By IID/non-IID***





***Data Dispersion***





***Local Round***





***Delayed Update – By Proportion***





***Delayed Update – By Speed***





***Share data***

