1. Introduction
2. Background

2.1 Machine Learning in Bioinformatic fields

What can be done using ML (several examples using Neural network), may other ways that use old machine learning.

2.2 Performance and dataset

If there’s a paper about data size and training performance… Introduce several famous datasets in biology/medical fields. But compare to large dataset/baseline like ImageNet xxx. Or describe the need for dataset like lack sth???

2.3 Federated Learning

What is / why use FedML (privacy/distribute computing or storing resources), challenges, federated learning frameworks. Federated success case in biology field.

1. Datasets and Methods

In this project, the CIFAR-10 dataset has been used for locally simulating different scenarios of federated learning, the OIA-DDR dataset 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.

We test 1000 epochs (for centralized training) or global rounds (for federated learning) for each simulation task and 100 epochs/rounds for the cloud implementation

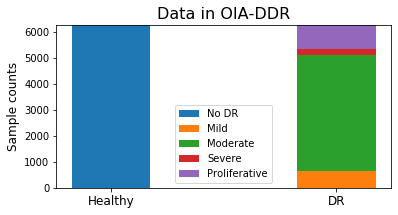
3.1 CIFAR-10 dataset

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>)



(Figure xxx each class’s count)

3.3 Training designs

The code is written in Python3 and TensorFlow 2.0.1 is used.

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 crossentropy.

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. The batch size is set to 300; loss will be calculated by binary crossentropy.

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

In this project, we evaluate the training performance with accuracy and loss. These functions are provided by TensorFlow.

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 worker nodes are involved in this simulation system. We assume no data or nodes will be dropped out or added in 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. Worker nodes load weight from the central node,
2. Worker nodes train the model over their local data and save the history,
3. Worker 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. Since the original results are fluctuated, we will only perform windowed accuracy and loss in this report. The original records are available in the GitHub repository.

***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.

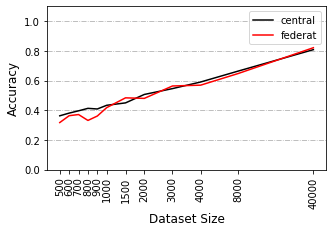
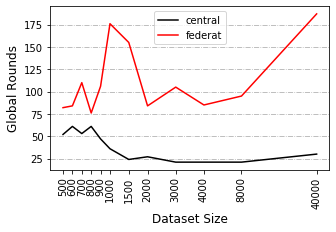
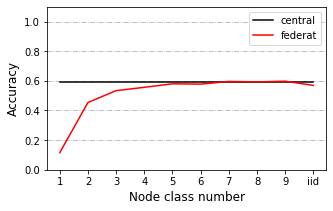
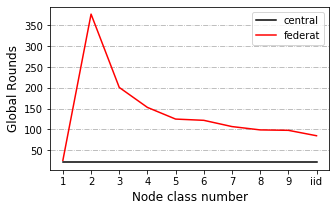
 

Figure xxx

The figure shows the training result with different size of dataset. (window size=20) 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, so we show it by identifying how many classes each node has. 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=20).

We also have observed different behaviors between tests using dataset of size 4000 and 40000 under 2 class non-IID circumstances.

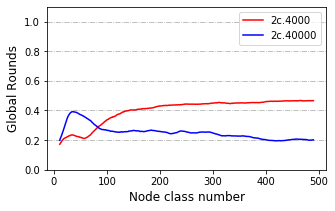


Figure xxx

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. Worker 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 within the central node. In each scenario, we change 1 parameter and compare it to the one without that change.

By default, we test each scenario under 2 classes, 5 classes and IID classes circumstances with dataset of size 4000 and under 2 classes, 3 classes, 5 classes and IID classes circumstances with dataset of size 40000.

***Byzantine Attacks***

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 dataset of size 4000 and 40000. Additional tests of 3% abnormal node size are performed with dataset of size 40000.

***Data Dispersion***

In some tests we distribute our data to 10 workers, in other tests to 40 workers. In this scenario, we compare the training performance between 10 worker and 40 worker cases.

***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***

This scenario is tested with 1 class, 2 classes, 5 classes and IID classes, we split the training set to 10 nodes. In pilot 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.

We apply this strategy to our model with sharing size equals to 2.5% of the total training set. In each node there will be a mixture of shared data and local data. In our design, an IID local training set is randomly resampled from the local sample pool. We compare the result of our design to the original strategy. In order to simplify the pipelines, pre-training step is not applied.

3.5 Cloud implementation (NOT yet finished)

This

1. Results

4.1 Scenarios testing

In this section we will show part of our testing results.

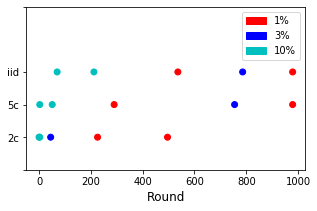
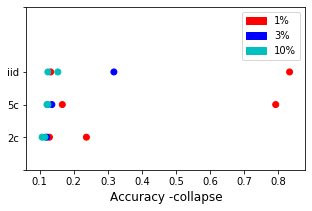
In this section we will show part of our testing results, the full results are available in the appendix. Though the model will overfit at early rounds, we plot the full line to show its trend.

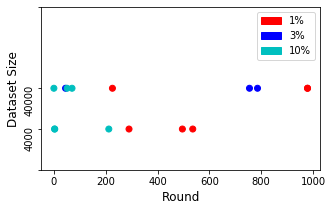
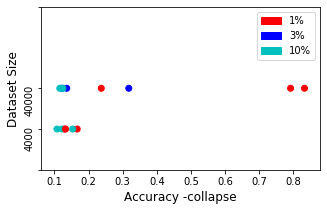
***Byzantine Attacks***

Acc at dropped – round index at dropped

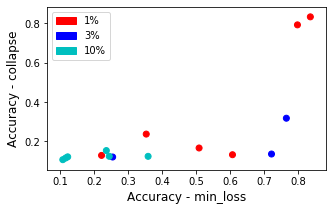
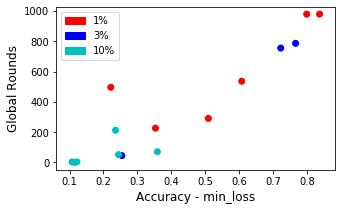
In pilot tests we have observed a slower convergence speed when the dataset is non-IID. Here

The collapsing speed



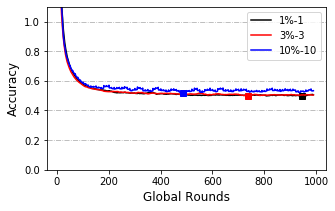
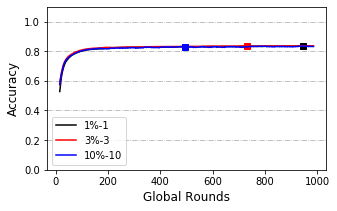


Acc\_at\_min\_loss – index\_drop/acc\_at\_drop

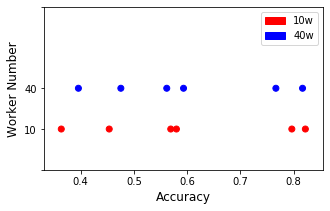
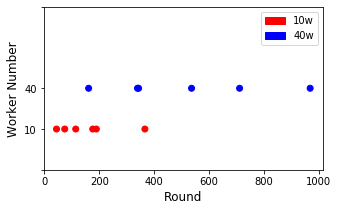


Treshhold0.025, window 21

If model’s original(? Not plotted here, maybe not so relative to it?) acc/ can reach to higher acc, it takes longer epochs to let its performance drop (drop slower). At final most will drop to low or random acc.



***Data Dispersion***



Left: round-index of min-loss, right-acc at min-loss-point

Apply paired t-test and get:

Accuracy: p-value = 0.7098641046040814, not big difference (cannot reject H0)

Converge round: p-value=0.00593646963137399

***Delayed Update – Proportion’s effect***

***Delayed Update – Speed’s effect***

***Share data***

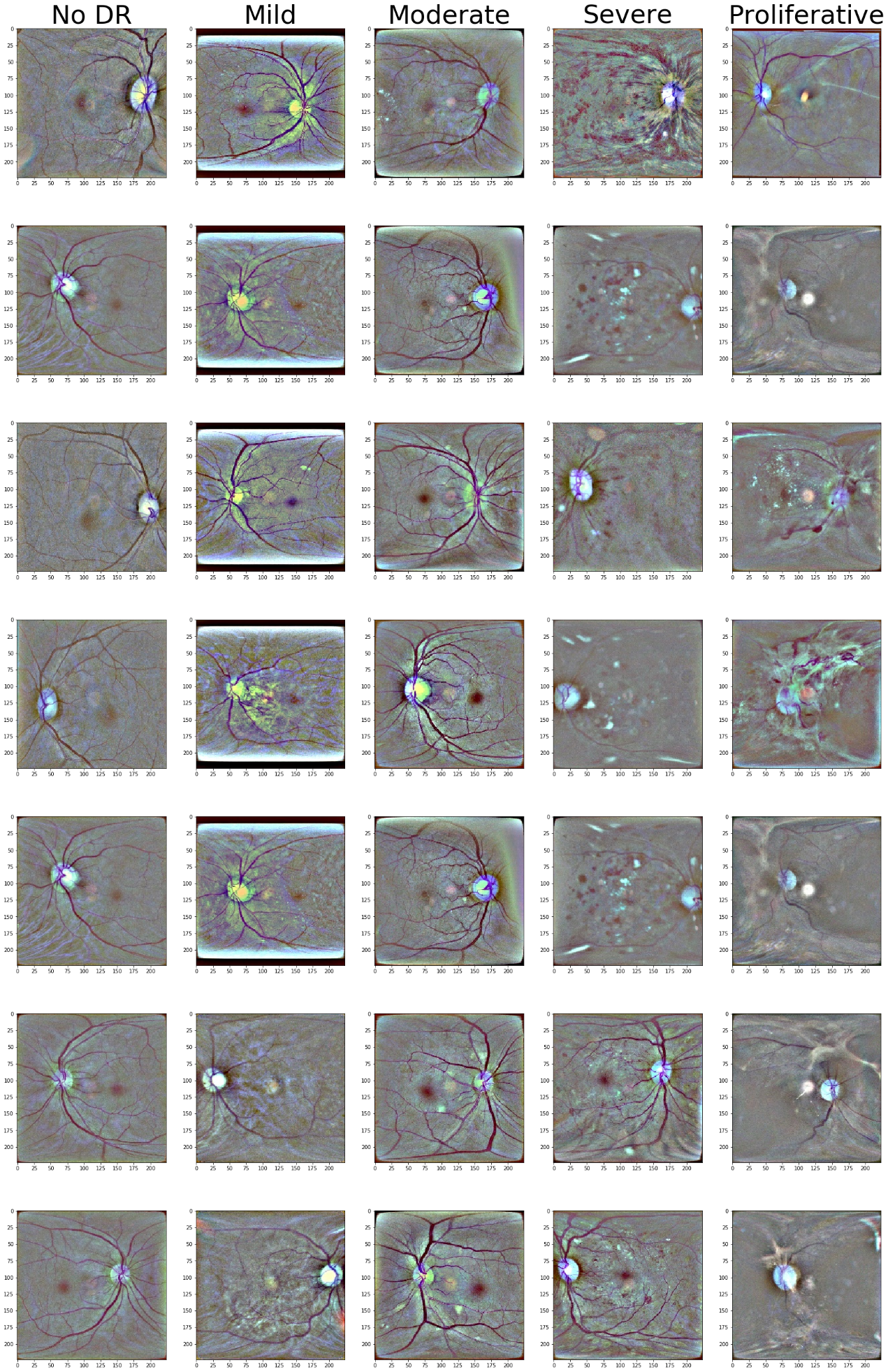
4.2 Cloud implementation

1. Discussion

(like how to evaluate on node.)

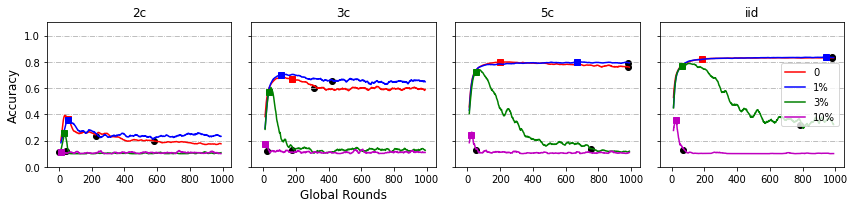
1. Conclusion
2. Acknowledgement
3. Reference

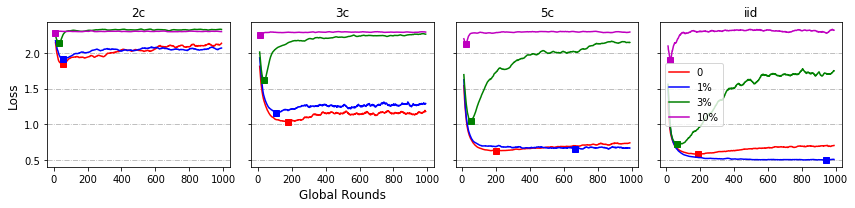
Appendix 1 OIA-DDR dataset after preprocessing



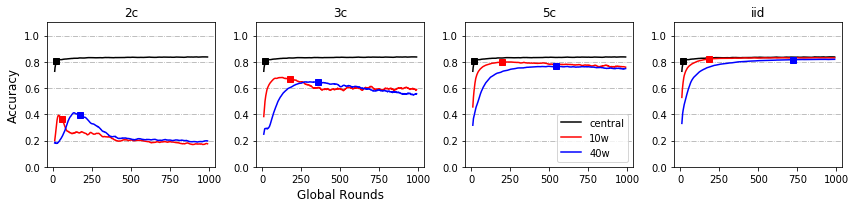
Appendix- Scenario testing results with 40k training set

***Byzantine Attacks***





***Data Dispersion***





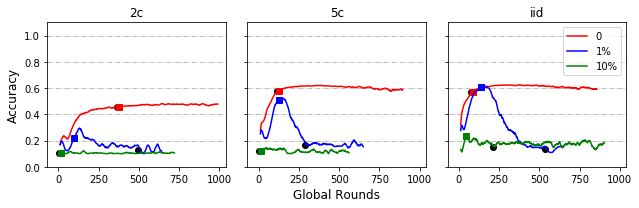
***Delayed Update – Proportion’s effect***

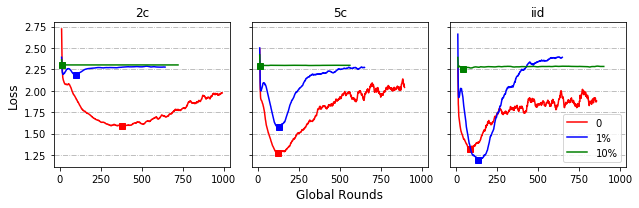
***Delayed Update – Speed’s effect***

***Share data***

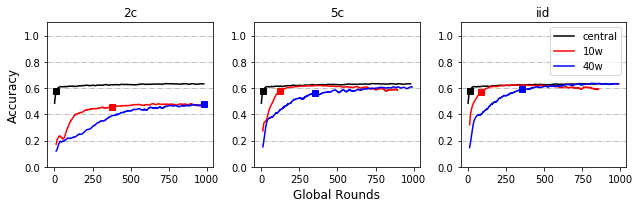
Appendix- Scenario testing results with 4k training set

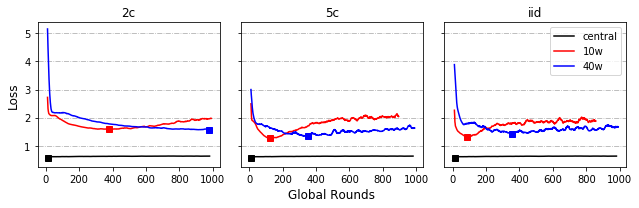
***Byzantine Attacks***





***Data Dispersion***





***Delayed Update – Proportion’s effect***

***Delayed Update – Speed’s effect***

***Share data***