Implementation of Federated Learning Emulation Framework Considering Resource Heterogeneity

Student	Farzad Mohammadi
Student ID	810100552
Course	Distributed Optimization and Learning

1 Summary

In this project, we developed a framework for emulation of Federated Learning (FL) algorithms using Message Passing Interface (MPI) that can be used for performance evaluation of various FL algorithms. In fact, we have demonstrated usability of this framework by implementing FedAvg and FedProx algorithms.

2 Problem Instance Description

2.1 Dataset description

- Dataset: Boston housing prices
- Number of samples: 506
- Number of features: 13

2.2 Neural Network (NN) description

- Architecture: Multi Layer Perceptron (MLP)
- Number of hidden layers: 2
- Activation functions: ReLU
- Modeling framework: pyTorch

3 Supplementary Files Description

- ***centralized.py***: This file contains centralized implementation of NN training.
- ***FedProx_sim.py***: This file contains implementation of FedProx and FedAvg simulation functions and utilities.
- ***protocol_demo.py***: This file presents a simple demo of our framework scaffolding that later is used for implementation of communication protocols used within FL training procedure.
- ***FedProx_emu.py***: This file contains implementation of FedProx and FedAvg emulation along with protocol communication between different processes.
- ***plot_res.py***: This simple script is used for plotting comparison plots, which is used within this report.
- ***utils.py***: This file includes a logger system implementation used within framework.

4 Centralized Learning

In this phase, we are just trying to get familiar with our dataset and implement some helping functions that can be used later for FL implementation.
First, we have designed an MLP corresponding to the description given in 2.2. We named this model as "CentralizedModel" and we will used it in the rest of this project without any changes. Then, we created a training pipeline using pyTorch SGD optimizer; pyTorch dataloader, which is used for mini-batch training; and L1loss function.
In order to report results, knowing that the loss value is actually a Random Variable (RV), we repeat entire training process for a number of times and then use their results to calculate 95% Confidence Interval. Centralized training result can be seen in figure below:

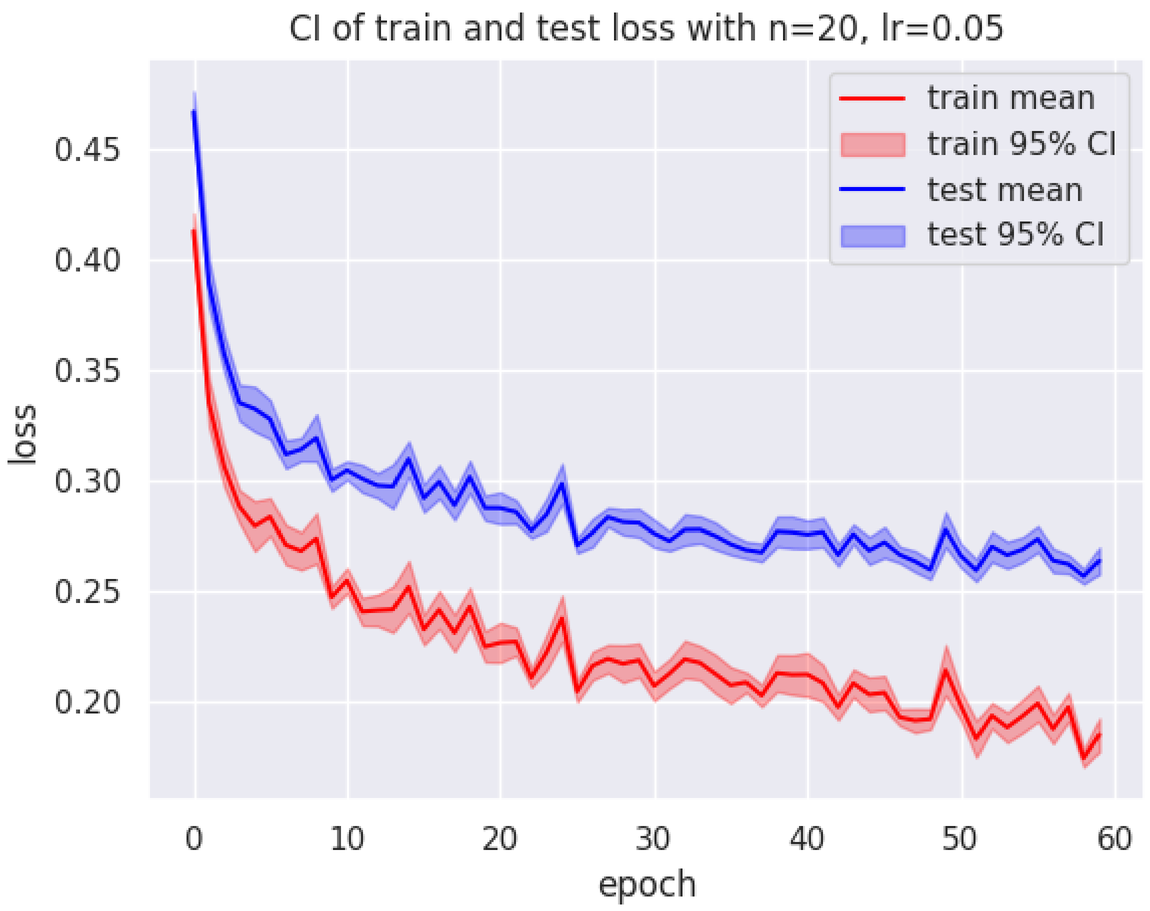

Figure 1: Centralized training result

Now that we have a basic understanding of our dataset and MLP, we can go on with implementing FL.

5 FedAvg and FedProx Simulation

5.1 *data_splitter* function

From the beginning we said that we are going to emulate (here we discuss the simulation but emulation will be elaborated in later sections) "Resource Heterogeneity" in FL. To do that, we have to synthetically generate heterogeneity that from now on we call it **non-iid** setup. This function is responsible for generating iid and non-iid datasets for agents participating in learning process.
Now we have to explain how we created non-iid datasets. First, we generate $n\_agents - 1$ random variables from uniform distribution (lower bound is zero and upper bound is $dataset\_size - 1$). Then, we insert 0 and $dataset\_size - 1$ to the list of generated random variables. Finally, we sort the resulting list and after that we have our data indices for our agents' dataset, which are non-iid.

5.2 Simulation

The $\boldsymbol{main}$ function implements the entire simulation procedure, from dispersing global model to agents, agents training procedure, and model aggregation. Note that there are only two differences between FedAbg and FedProx simulation:

- Inexact proximity term: This term is calculated in $\boldsymbol{agent\_trainer}$ function and is governed by $\boldsymbol{mu}$ parameter, which is an input to the FedProx.
- Adaptive epoch number: This feature helps to materialize inexact solution of optimization that is solved by agents in every training round. This feature is governed by Boolean variable $\boldsymbol{variable\_E}$ that is also an input to the algorithm.

Epoch number adaptivity is controlled by a simple algorithm: $n\_epoch = (size\_of\_data\_set)/100$. This simple algorithm makes sure that an agent having a large number of data samples will train more, and on the other hand, an agent having limited number of data samples will train less. This behavior is exactly what that was desired in order to face resource heterogeneity challenge, which is part of FedProx algorithm.

5.3 Simulation Results

In this section we present simulation results. Note that in all simulations we have three agents and one coordinator performing the training process. The first figure in this section depicts FedAvg's performance in iid setting:

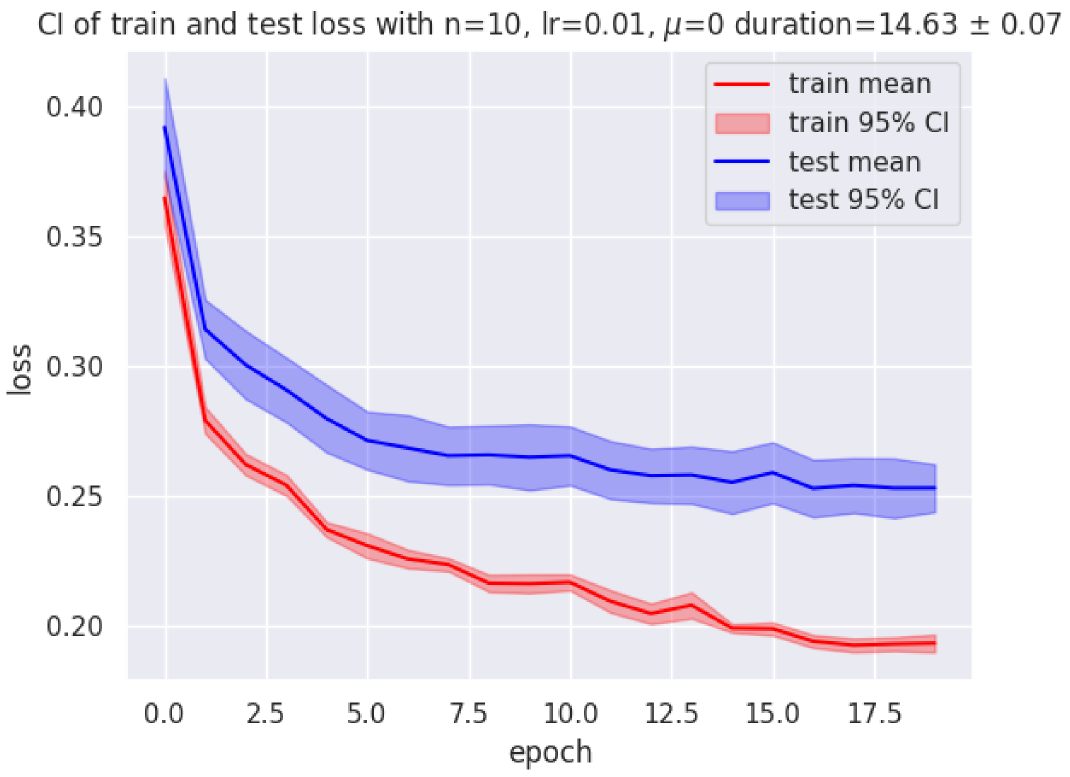

Figure 2: FedAvg's simulation in iid setting

Note that in this figure we see that mu is zero, which is expected, also we see algorithms runtime with 95% confidence interval. We will talk about runtimes and their comparison and analysis in later sections.
Next figure shows FedAvg in non-iid setting:

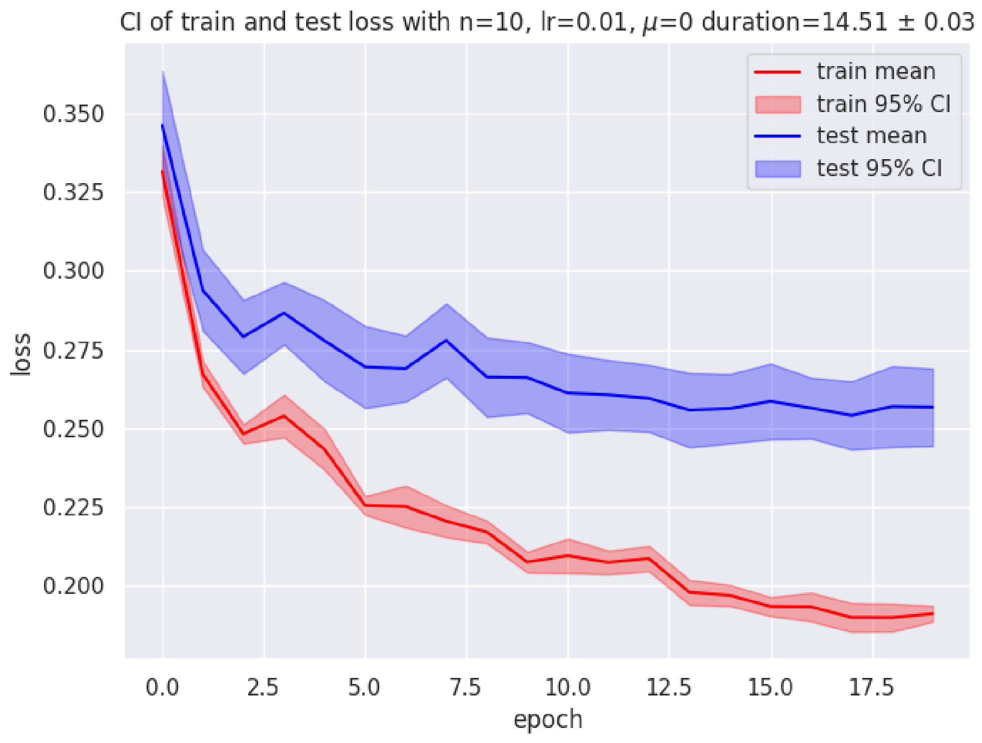

Figure 3: FedAvg's simulation in non-iid setting

Note that in this figure CI seems to be larger than iid setting that is plausible.
Now we see FedProx in iid setting:

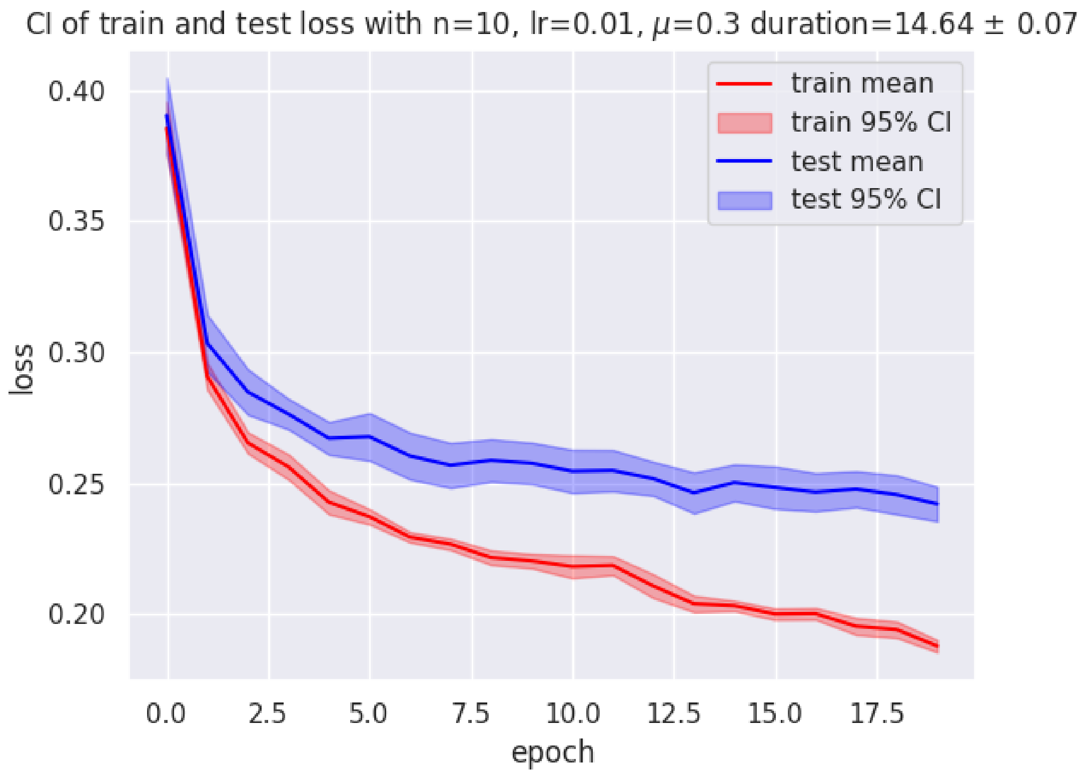

Figure 4: FedProx's simulation in iid setting

Comparing to FedAvg in iid setting we see that CI range is tigher and this feature can be attributed to the proximity term in FedProx algorithm.

Finally, FedProx performance in non-iid setup:

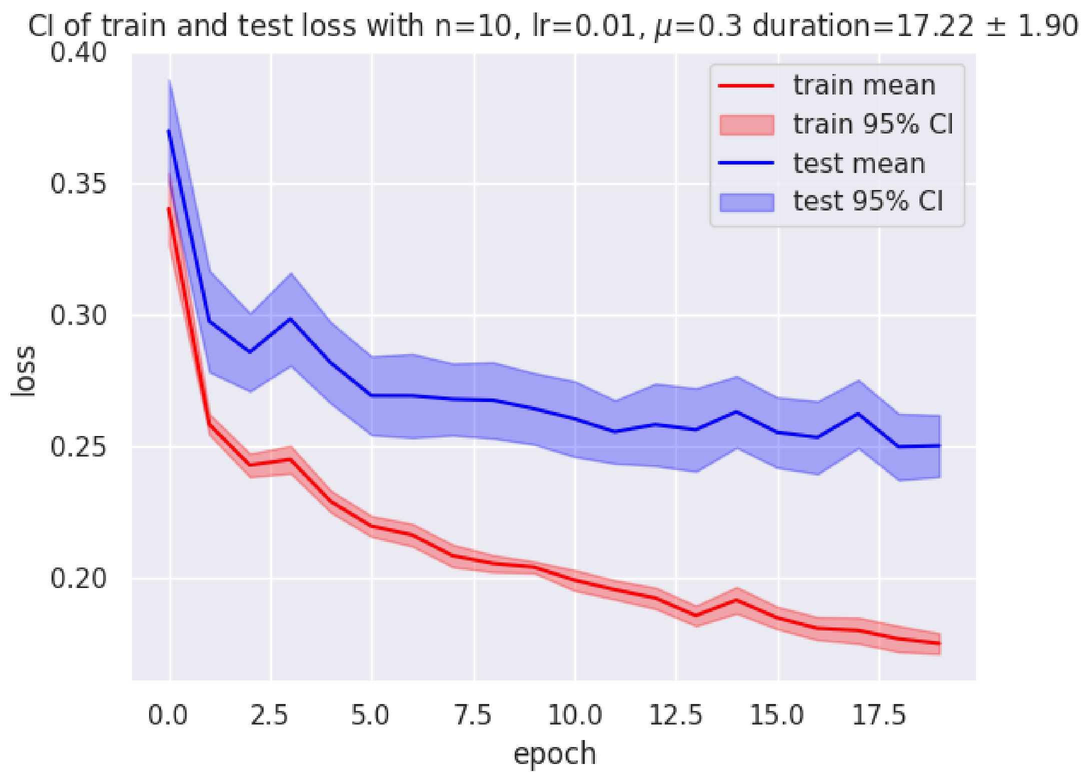

Figure 5: FedProx's simulation in non-iid setting

Again, we see that CI range gets larger regardless of the algorithm when we switch to non-iid setting. We will compare FedAvg performance with FedProx's in later sections.

6 Communication Protocol and Distributed Computing Infrastructure

6.1 Distributed Computing Infrastructure

In this projects we have used Message Passing Interface to implement a simple distributed computing infrastructure. We simply could have used pyTorch Distributed, but we wanted gain further understanding of how frameworks like mentioned one work under the hoods (actually pyTorch Distributed can use MPI as it's backend among few options available). This framework uses message passing method to exchange information between different processes, which is on a single node or cluster of nodes. An important question is why we haven't used multiprocess package from python's standard library?. The answer is that despite the fact that both perform the same in single node settings, multiprocess package doesn't offer any solution for cluster of computing nodes. Nevertheless, using MPI, developers can easily scale their application to a cluster of nodes in a LAN.

6.2 Communication Protocol

Although MPI provides means to synchronize processes or communicating tools between them, we, as developers, still have to use provided APIs to implement our desired logic. In this project we have implemented a protocol very similar to the one shown in the figure below.

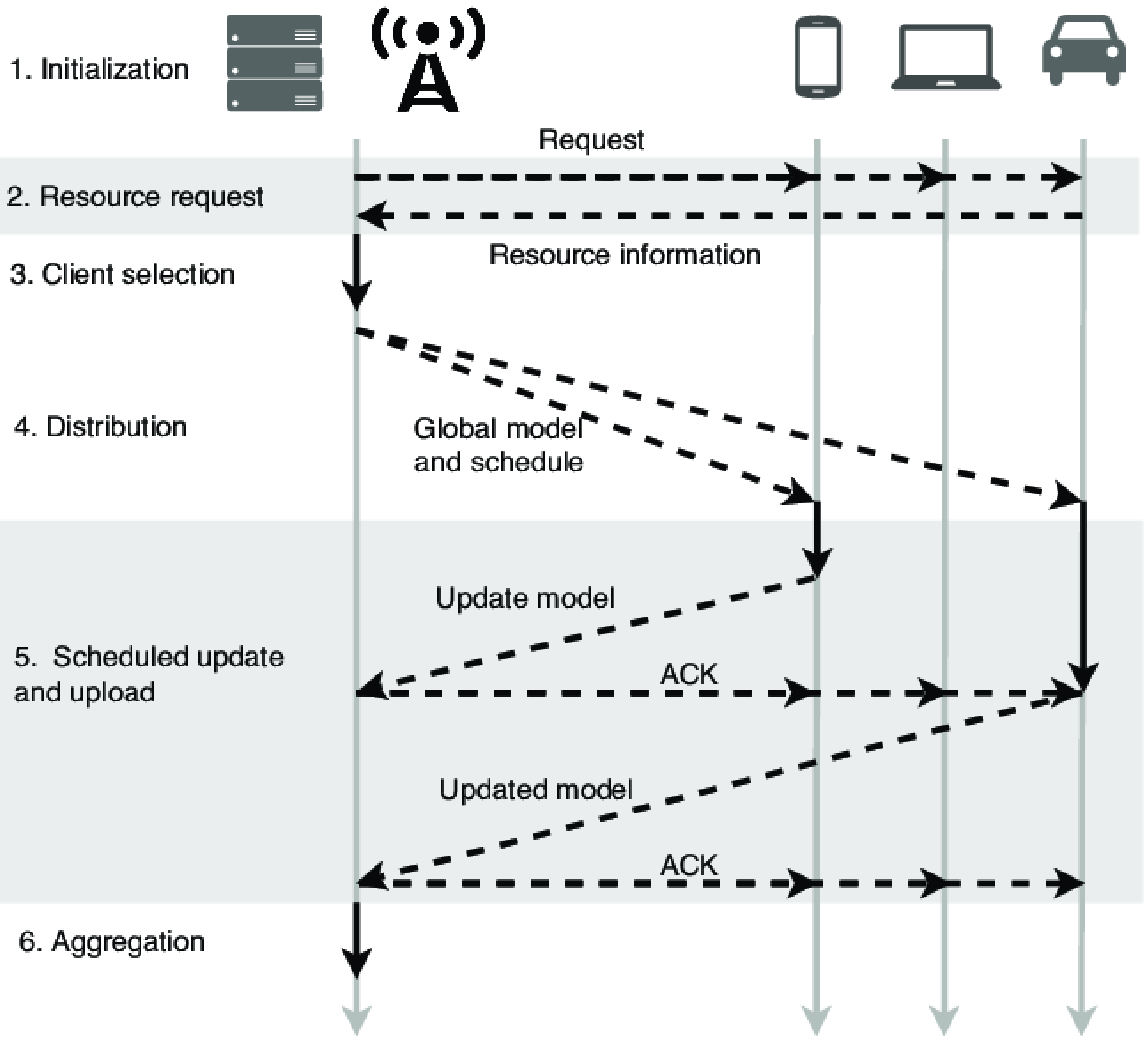

Figure 6: FedCS protocol procedure from Nishio, Takayuki, and Ryo Yonetani. "Client selection for federated learning with heterogeneous resources in mobile edge." ICC 2019-2019 IEEE international conference on communications (ICC). IEEE, 2019.

The only difference between our implemented protocol and the one that is shown is we haven't implemented ACKs. We have provided a demo file for this projects that can be illuminating.

7 FedProx and FedAvg Emulation

Using communication protocol and utility functions, which we introduced them in previous sections, emulation framework of FedProx and FedAvg is developed. Without any further discussion on the implementation details, we move on to present results. In the first figure we can see FedProx emulation in iid setting.

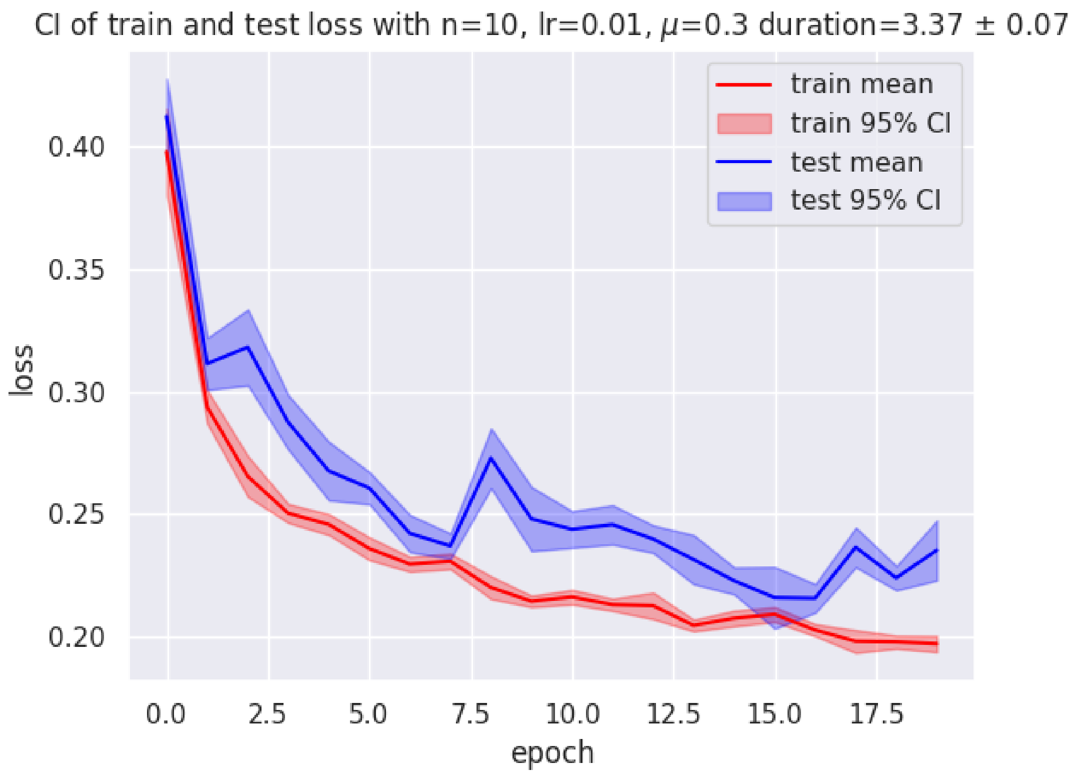

Figure 7: FedProx emulation in iid setup

Note the substantial difference in runtime comparing to simulation runtimes. Next we see FedProx emulation in non-iid setting.

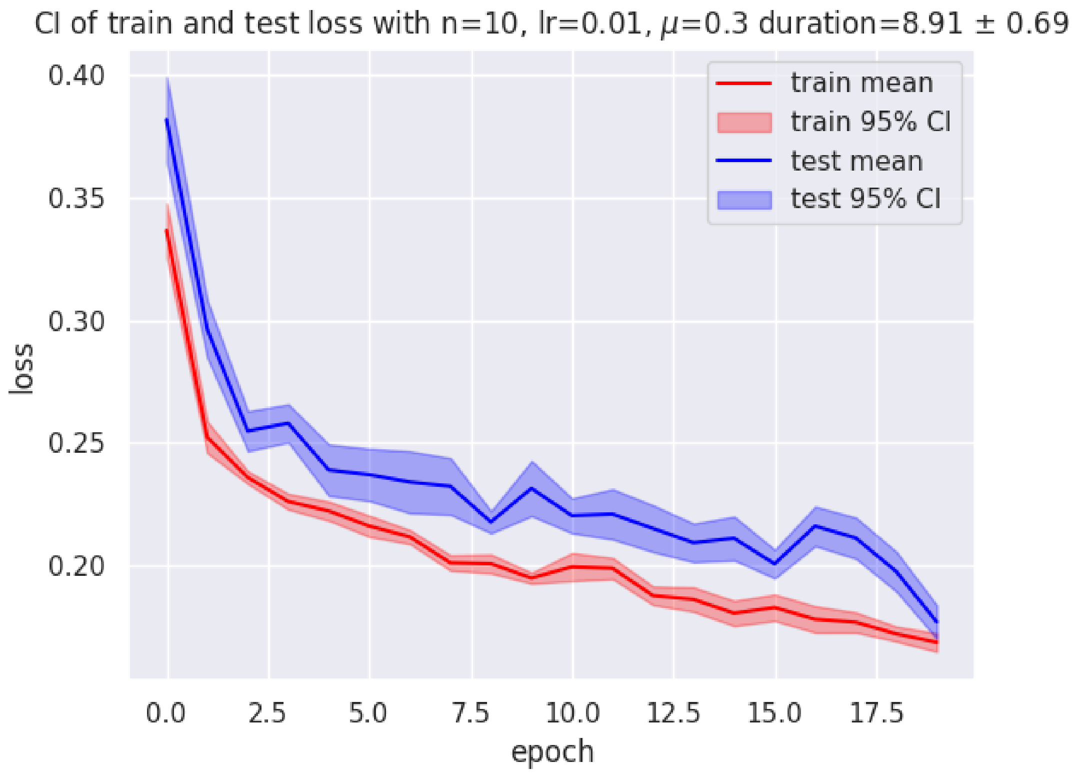

Figure 8: FedProx emulation in non-iid setup

Note the huge gap between runtimes of FedProx emulation in iid and non-idd setup. We will explain the underlying reasons for this result in the next section.

8 Comparisons and Analysis

8.1 Loss Comparison and Analysis

In this section we compare FedAvg and FedProx performances in terms of their loss value. First we start with iid setup.

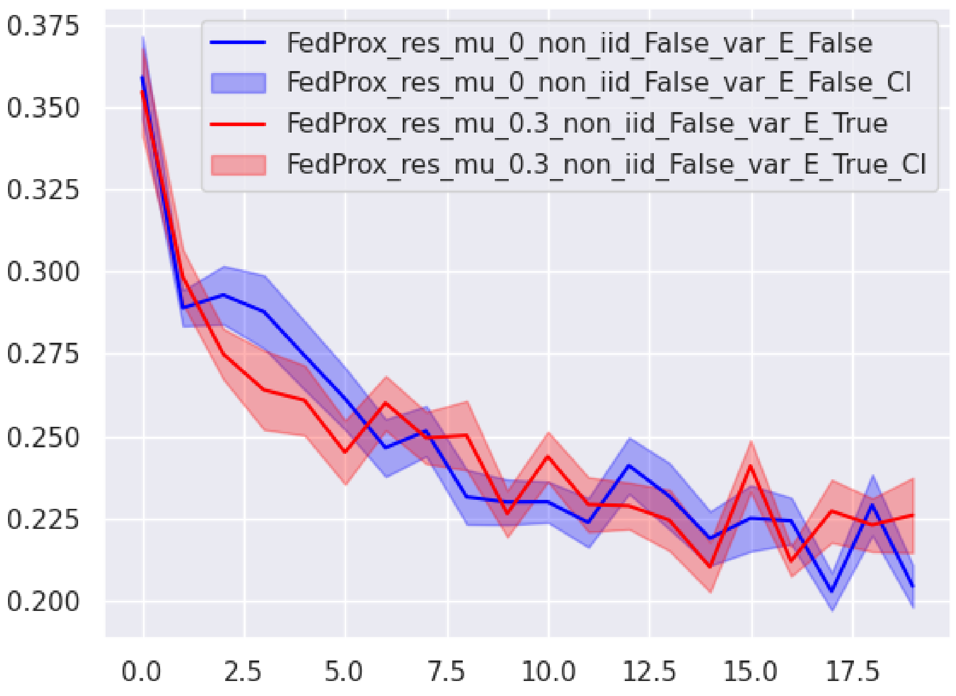

Figure 9: FedAvg and FedProx comparison in iid setup

The blue color corresponds to FedAvg algorithm and the red color to FedProx. We see that in ***iid*** setting there is not much difference in two algorithms' performance. To explain why this happens, let's remind ourselves what was the primary purpose of FedProx algorithm: Handling resource heterogeneity challenge. But, in iid setup we don't have any resource heterogeneity. Consequently, it is reasonable to see not much difference in performance. In iid setup, adaptive epoch algorithm has absolutely no effect and results in constant epochs for every agent, which is the same as FedAvg's procedure.
Next we see comparisons in non-iid setup.

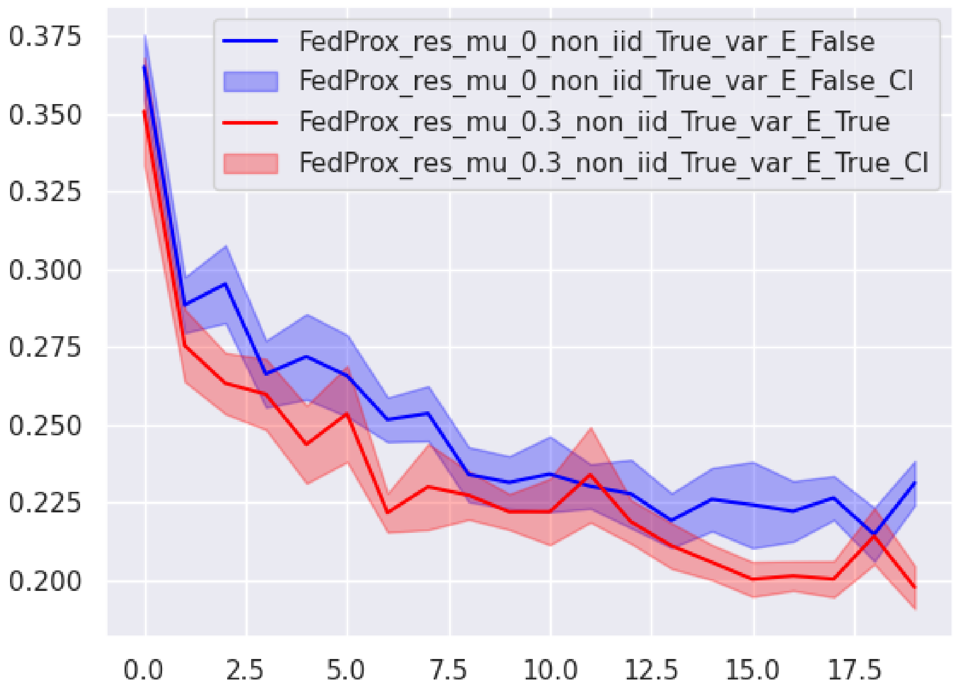

Figure 10: FedAvg and FedProx comparison in non-iid setup

In ***non-iid*** settings, we see that FedProx has outperformed FedAvg considerably. There are two factors that have contributed to this result. First, adaptive epoch algorithm advocates for more training of agents having more data samples. Second, proximity term urges the optimization process to converge to a global model and avoid further divergence.

8.2 Runtime Comparison and Analysis

Runtime Comparison			
setup	FedAvg Simulation	FedProx Simulation	FedProx Emulation
iid	14.63 ± 0.03	14.64 ± 0.07	3.37 ± 0.07
non-iid	14.51 ± 0.03	17.22 ± 1.9	8.91 ± 0.69

There are some interesting points about results that we have to explain.
First, emulation runtimes are substantially lower than simulation runtimes. To understand this, we have to explain what factors contributes to overall latency in each setups. In simulation, coordinator and all three agents computations are being performed in a single process in a ***sequential*** manner. So the overall delay in simulation setup is sum of every participants' (coordinator and agents) delay. On the other hand, in emulation setting, coordinator and agents perform their computations in different processes in ***parallel*** manner. As a result, the total delay in emulation setting is the sum of coordinator delay plus the delay of slowest agent's delay. But, in emulation setting we have to factor in another important factor that helps us to explains why the runtime of FedProx emulation is much higher in non-iid setting than iid. The answer is ***communication and synchronization overhead***.
To put it simply, think about this situation: One the agents, agent A, has 80 percent of the data and the other two agent have 20% remaining data samples. As we said before, FedProx assigns more number of epochs for the agents having more data. So, agent A needs more time for training, but this means at every round when other two agents done with training, they have to wait for agent A to finish so that coordinator starts new training round. As a result, the total runtime increases in these situations.