University of Tehran

Faculty of Engineering

School of Electrical and Computer Engineering

Distributed Optimization and Learning

Project 1

Federated Learning for Image Classification: FedAvg, FedProx, and ADMM-Based Algorithms with IID and Non-IID Data Distributions

Hafez Ghaemi

810199239

Fall 2022

Abstract

In this project, we will consider the problem of federated learning (FL) for image classification. The federated variant we consider is horizontal (sample-based), i.e., the feature space is shared among agents, while their samples are different. Each agent trains its own local model, and sends its local weights to the coordinator who builds the global and aggregated model. We implement three different federated algorithms; federated averaging (FedAvg), FedProx, and Linearized inexact ADMM-based federated learning (LIADMM). We train these algorithms using both iid and non-iid distributions of distributed Fashion MNIST dataset, and compare their validation performance with the centralized training paradigm.

1. Introduction

Horizontal (sample-based) federated learning is a paradigm for distributed optimization, in which a set of agents who share a common feature space but possess different data samples, collaborate to optimize an objective function. Due to the importance of privacy and data protection, the agents only share a gradient with a controller who is in charge of aggregating these gradients and build a global model [1]. In this project, we consider a special case of federated learning, that is image classification using convolutional neural networks. The network architecture is the same for all agents so that the controller can easily aggregate the weights of each model.

One of the main challenges of federated learning is the statistical heterogeneity of the agents' datasets. The assumption that the agents' datasets are all independent and identically distributed (iid), is very strong for real-world applications involving various clients [2]. The initial variant of federated learning (federated averaging or FedAvg), does not perform well in the presence of data heterogeneity. To address this issue, another algorithm, called FedProx [2], has been proposed that introduces a regularization term to ensure that the weights of the local models do not get too far from the weights of the global model. In addition to centralized training, FedAvg, and FedProx, we will also implement an ADMM-based algorithm, called linearized inexact ADMM-based federated learning (LIADMM) [3]. This method has been tested before on linear regression and logistic regression problems, and we use it in a layer-wise fashion to optimize the loss function of a deep neural network. The ADMM-based algorithms for federated learning have been shown to converge with a linear rate even when trained in a communication-efficient way, which makes it a good candidate for future applications [3].

We will train all the models on the Fashion MNIST [4] benchmark for both iid and non-iid data distributions. The network used is a convolutional neural network that is typically used for image classification. In section 2, we will discuss the Fashion MNIST dataset. Section 3 lays out the network architecture. The three implemented algorithms are explained in Section 4, and the evaluation results are given in Section 5. Finally, Section 6 offers a brief discussion around the results and concludes this report.

2. Dataset

The dataset we used for training and evaluation is Fashion MNIST [4], which consists of 60000 training samples alongside a test set with 10000 samples. The dataset consists of ten classes of clothing items and it is completely balanced in terms of class distribution. Figure 1 shows a sample of each class belonging to this dataset.

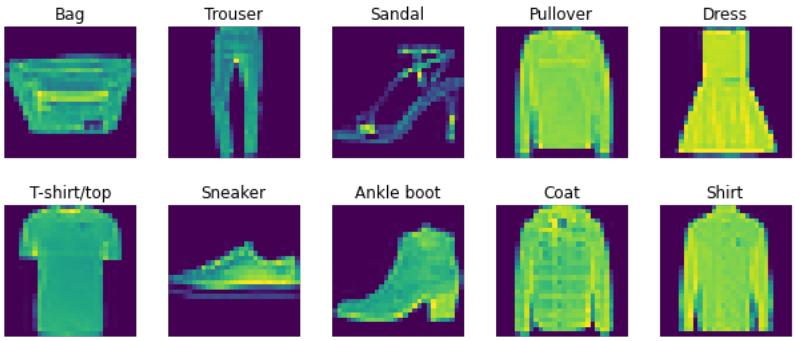


Figure 1. Class samples of the Fashion MNIST dataset

For iid splits, we shuffle the dataset and randomly sample the required fraction of each dataset. For non-iid splits, we split each class into shards, and for each split randomly select data shards from only two random classes.

3. Network Architecture

We use the same convolutional neural network architecture as proposed by McMahan et al. [1] in the original federated learning paper.

Layer	Dimension	Properties
Input	(B, 1,28,28)	
Convolution	(B, 32,26,26)	Padding = 1, Stride = 1 Kernel Size = 5, Activation = ReLU
Max Pooling	(B, 32,14,14)	Padding = 1, Stride = 1, Kernel Size = 2
Convolution	(B, 64, 12, 12)	Padding = 1, Stride = 1 Kernel Size = 5, Activation = ReLU
Max Pooling	(B, 64,7,7)	Padding = 1, Stride = 1, Kernel Size = 2
Flatten	(B, 3136)	
Dense	(B, 512)	Activation = ReLU
Dense	(B, 10)	Activation = Softmax

The total number of trainable parameters in the model is 1,663,370. We set batch size to 128 and the number of training epochs to 10, for both centralized training and the local models in federated learning. The optimization algorithm used in all models is stochastic gradient descent with a learning rate of 0.05 and momentum equal to 0.9.

4. Methodology

4.1. Federated Averaging (FedAvg)

The first implemented algorithm is federated averaging which is the original variant of federated learning proposed by McMahan et al. [1]. In our case, the goal is to optimize the loss function of a neural network with weights w, which is the result of aggregation over the distributed agents' loss functions,

$$\min_{w \in \mathbb{R}^d} f(w) \qquad \text{where} \qquad f(w) \overset{\text{def}}{=} \frac{1}{n} \sum_{i=1}^{n} f_i(w).$$

Assuming that the data is partitioned into k distributed clients with different number of data samples, we may rewrite the cost function as

$$f(w) = \sum_{k=1}^{K} \frac{n_k}{n} F_k(w) \quad \text{where} \quad F_k(w) = \frac{1}{n_k} \sum_{i \in \mathcal{P}_k} f_i(w).$$

A server is in charge of aggregating the agents' parameters. The pseudocode of the federated averaging (FedAvg) is given below,

Algorithm 1 `FederatedAveraging`. The K clients are indexed by k; B is the local minibatch size, E is the number of local epochs, and η is the learning rate.

Server executes:
  initialize w_0
  for each round $t = 1, 2, \dots$ **do**
    $m \leftarrow \max(C \cdot K, 1)$
    $S_t \leftarrow$ (random set of m clients)
    for each client $k \in S_t$ **in parallel do**
      $w_{t+1}^k \leftarrow \text{ClientUpdate}(k, w_t)$
    $w_{t+1} \leftarrow \sum_{k=1}^{K} \frac{n_k}{n} w_{t+1}^k$

ClientUpdate(k, w): *// Run on client k*
  $\mathcal{B} \leftarrow$ (split $\mathcal{P}_k$ into batches of size B)
  for each local epoch i from 1 to E **do**
    for batch $b \in \mathcal{B}$ **do**
      $w \leftarrow w - \eta \nabla \ell(w; b)$
  return w to server

4.2. FedProx

The FedProx algorithm was proposed to address the non-iid data distribution among the clients. In this algorithm, a regularization term is introduced to ensure that the local updates do not become too distant from the global weights at each round which is likely to happen when we have heterogeneous datasets [2]. The pseudocode of this algorithm is given below,

Algorithm 2 `FedProx` (Proposed Framework)

Input: K, T, μ, γ, w^0, N, p_k, $k = 1, \cdots, N$
for $t = 0, \cdots, T-1$ **do**
  Server selects a subset S_t of K devices at random (each device k is chosen with probability p_k)
  Server sends w^t to all chosen devices
  Each chosen device $k \in S_t$ finds a w_k^{t+1} which is a γ_k^t-inexact minimizer of: $w_k^{t+1} \approx \arg\min_w \; h_k(w;\, w^t) = F_k(w) + \frac{\mu}{2}\|w - w^t\|^2$
  Each device $k \in S_t$ sends w_k^{t+1} back to the server
  Server aggregates the w's as $w^{t+1} = \frac{1}{K}\sum_{k \in S_t} w_k^{t+1}$
end for

4.3. Linearized inexact ADMM-based federated learning (LIADMM)

A new set of federated learning algorithms has utilized ADMM for federated optimization [3]. Below, you can find the main version of these algorithms,

Algorithm 2: ADMM-based federated learning

Initialize $\mathbf{x}_i^0, \boldsymbol{\pi}_i^0, \sigma_i > 0, i \in [m]$. Set $k \Leftarrow 0$.
for $k = 0, 1, 2, \ldots$ **do**
  Weights upload: Each client sends the parameters $\mathbf{x}_i^k$ and $\boldsymbol{\pi}_i^k$ to the central server.
  Global aggregation: The central server calculates the average parameter $\mathbf{x}^{k+1}$ by

$$\mathbf{x}^{k+1} \;=\; \text{argmin}_{\mathbf{x}}\; \mathcal{L}(\mathbf{x}, X^k, \Pi^k). \tag{2.16}$$

  Weights feedback: The central server broadcasts the parameter $\mathbf{x}^{k+1}$ to every local client.
  for $i = 1, 2, \ldots, m$ **do**
    Local update: Each client updates its parameters locally and in parallel by

$$\begin{aligned} \mathbf{x}_i^{k+1} &= \text{argmin}_{\mathbf{x}_i}\; L(\mathbf{y}^{k+1}, \mathbf{x}_i, \boldsymbol{\pi}_i^k), \\ \boldsymbol{\pi}_i^{k+1} &= \boldsymbol{\pi}_i^k + \sigma_i(\mathbf{x}_i^{k+1} - \mathbf{y}^{k+1}). \end{aligned}$$

  end
end

where 2.16 is

$$\mathbf{x}^{k+1} = \text{argmin}_{\mathbf{x}}\ \mathcal{L}(\mathbf{x}, X^k, \Pi^k) = \sum_{i=1}^m \frac{\sigma_i \mathbf{x}_i^k}{\sigma} + \sum_{i=1}^m \frac{\boldsymbol{\pi}_i^k}{\sigma},$$

with

$$\sigma := \sum_{i=1}^m \sigma_i.$$

In a sense, a conventional ADMM algorithm is run among each agent and the controller, however, the update of the dual variable π_i is done locally by each agent. The main advantage of these algorithms is that they have theoretical convergence guarantees for periodic communications between the client and the server, and hence, they are communication-efficient. This communication-efficient algorithm is given below,

Algorithm 3: `CEADMM`: Communication-efficient ADMM-based federated learning

Initialize $\mathbf{x}_i^0, \boldsymbol{\pi}_i^0, \sigma_i > 0, i \in [m]$, an integer $k_0 > 0$. Set $k \Leftarrow 0$.

for $k = 0, 1, 2, \ldots$ **do**

  if $k \in \mathcal{K} := \{0, k_0, 2k_0, 3k_0, \ldots\}$ **then**

    Weights upload: Each client sends its parameters $\mathbf{x}_i^k$ and $\boldsymbol{\pi}_i^k$ to the central server.

    Global aggregation: The central server calculates the average parameter $\mathbf{x}^{k+1}$ by

$$\mathbf{x}^{k+1} = \sum_{i=1}^m \frac{\sigma_i \mathbf{x}_i^k}{\sigma} + \sum_{i=1}^m \frac{\boldsymbol{\pi}_i^k}{\sigma}. \tag{3.1}$$

    Weights feedback: The central server broadcasts the parameter $\mathbf{x}^{k+1}$ to every client.

  end

  for $i = 1, 2, \ldots, m$ **do**

    Local update: By letting

$$\mathbf{y}^{k+1} := \mathbf{x}^{\tau_k + 1}, \qquad \text{where } \tau_k := \lfloor k/k_0 \rfloor k_0,$$

    each client update its parameters locally and in parallel via solving

$$\mathbf{x}_i^{k+1} = \text{argmin}_{\mathbf{x}_i}\ w_i f_i(\mathbf{x}_i) + \langle \mathbf{x}_i - \mathbf{y}^{k+1}, \boldsymbol{\pi}_i^k \rangle + \tfrac{\sigma_i}{2} \|\mathbf{x}_i - \mathbf{y}^{k+1}\|^2, \tag{3.2}$$

$$\boldsymbol{\pi}_i^{k+1} = \boldsymbol{\pi}_i^k + \sigma_i(\mathbf{x}_i^{k+1} - \mathbf{y}^{k+1}). \tag{3.3}$$

  end

end

The optimization problem in the local updates of these algorithms does not always admit a closed form solution. Therefore, the authors proposed two inexact variants of these ADMM-based algorithm. Here, we use the linearized inexact ADMM-based federated learning algorithm (there is also a communication-efficient version),

Algorithm 4: LIADMM: Linearised inexact ADMM-based federated learning

Initialize $\mathbf{x}_i^0, \boldsymbol{\pi}_i^0$, a step size $\gamma > 0$. Set $k \Leftarrow 0$.

for $k = 0, 1, 2, \ldots$ **do**

Weights upload: Each client sends its parameters $\mathbf{x}_i^k$ and $\boldsymbol{\pi}_i^k$ to the central server.

Global aggregation: The central server calculates the average parameter $\mathbf{x}^{k+1}$ by

$$\mathbf{x}^{k+1} = \sum_{i=1}^{m} w_i \mathbf{x}_i^k + \gamma \sum_{i=1}^{m} \boldsymbol{\pi}_i^k. \tag{4.4}$$

Weights feedback: The central server broadcasts the parameter $\mathbf{x}^{k+1}$ to every local client.

for $i = 1, 2, \ldots, m$ **do**

Local update: Each client update its parameters locally and in parallel by

$$\mathbf{x}_i^{k+1} = \mathbf{x}^{k+1} - \gamma \nabla f_i(\mathbf{x}^{k+1}) - \tfrac{\gamma}{w_i} \boldsymbol{\pi}_i^k, \tag{4.5}$$

$$\boldsymbol{\pi}_i^{k+1} = \boldsymbol{\pi}_i^k + \tfrac{w_i}{\gamma}(\mathbf{x}_i^{k+1} - \mathbf{x}^{k+1}). \tag{4.6}$$

end

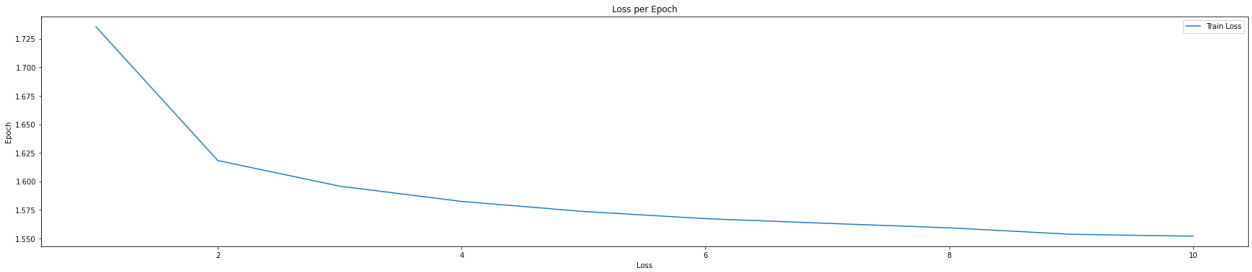
end

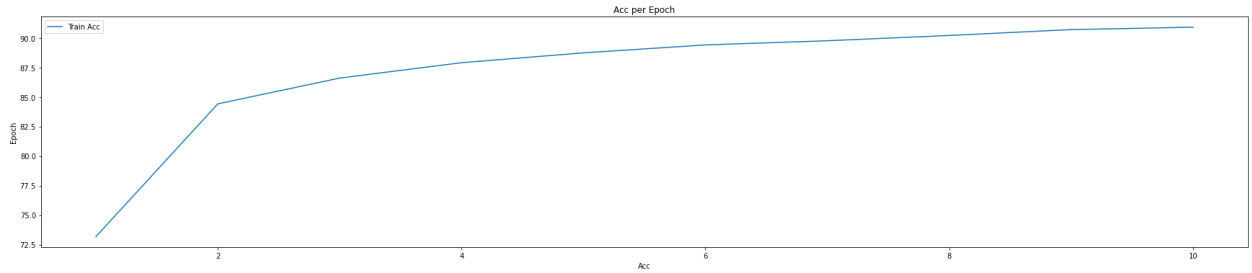
It should be noted that the numerical benchmarks for these algorithms were linear and logistic regression. To adopt them to neural network, we apply this algorithm in a layer-wise fashion with x as the weight matrix of the network, and π as an auxiliary (dual) weight matrix. According to 4.5., we set γ equal to the learning rate of the neural network.

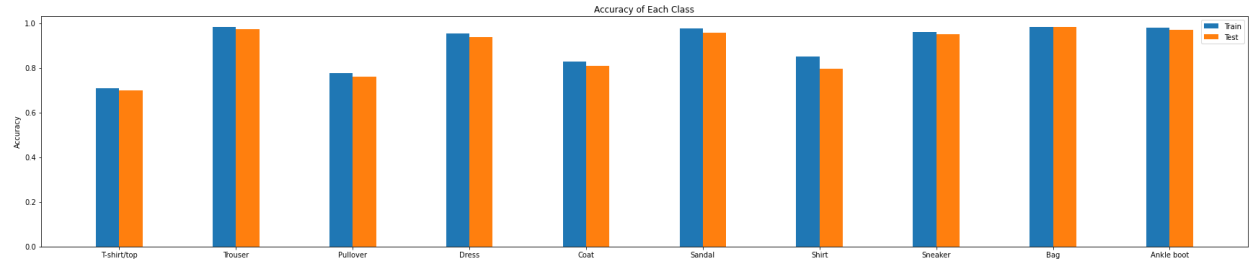
5. Results

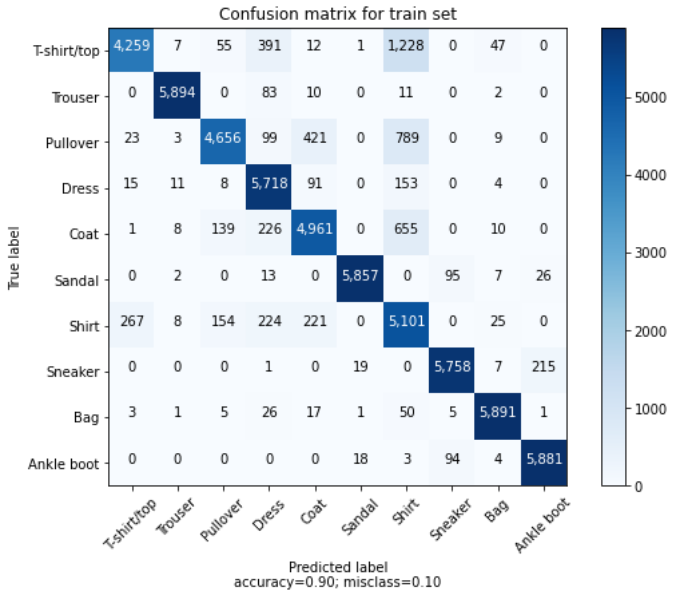
5.1. Centralized Training

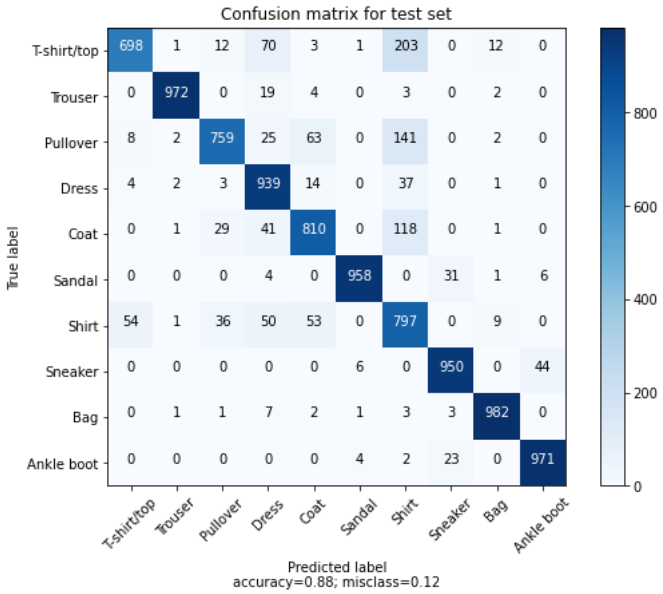
For centralized training, the hyperparameters are the ones mentioned in Section 3. Batch size is set to 128 and the number of training epochs to 10. The optimization algorithm used is stochastic gradient descent with a learning rate of 0.05 and momentum equal to 0.9. The results are given below,


Loss per Epoch
Train Loss
Epoch
Loss


Acc per Epoch
Train Acc
Epoch
Acc

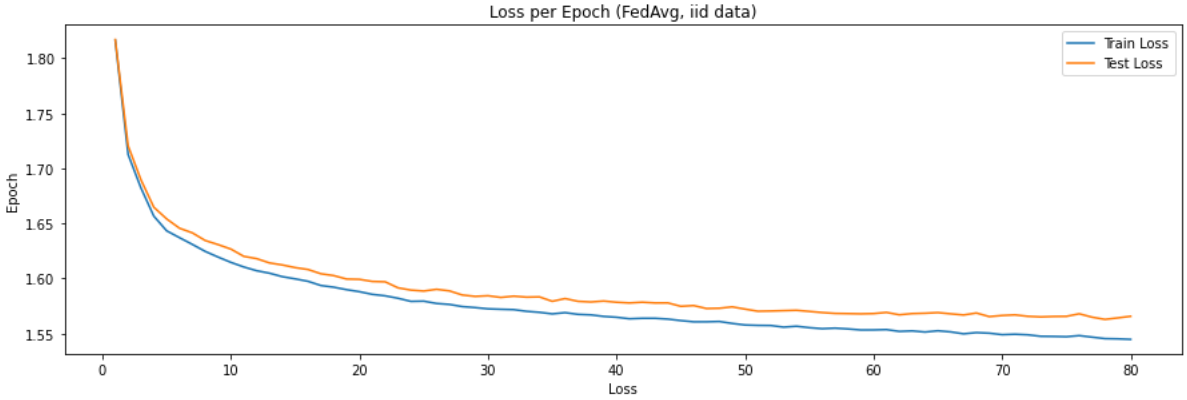

Accuracy of Each Class
Train
Test
Accuracy
T-shirt/top
Trouser
Pullover
Dress
Coat
Sandal
Shirt
Sneaker
Bag
Ankle boot

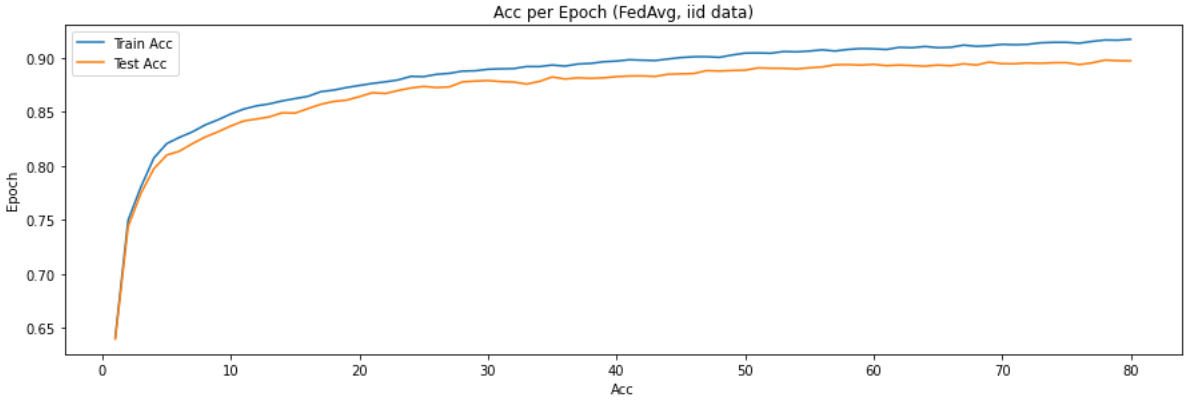

Confusion matrix for train set
True label
Predicted label
accuracy=0.90; misclass=0.10

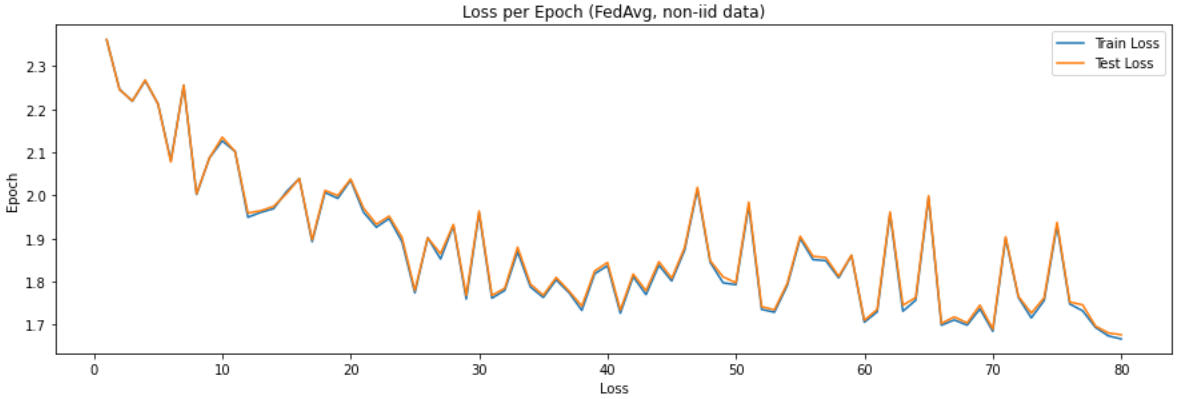


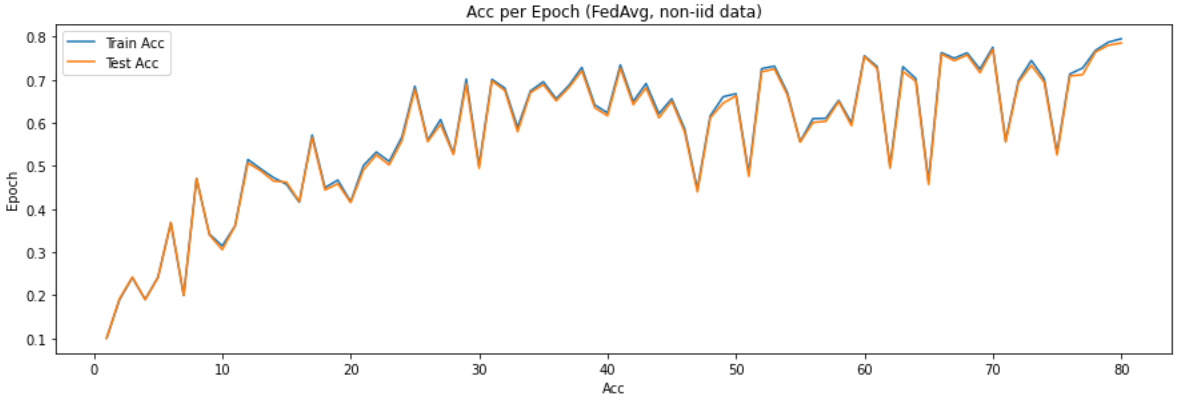
5.2. FedAvg

For FedAvg, apart from the common hyperparameters mentioned in Section 3, we only need to determine the number of rounds (E) which we choose as 80, and the fraction of clients chosen at each round (k) which we set to 0.1. The train and test loss accuracy for the global model at each round are plotted below for both iid and non-iid distributions,


Loss per Epoch (FedAvg, iid data)
Train Loss
Test Loss
Epoch
Loss

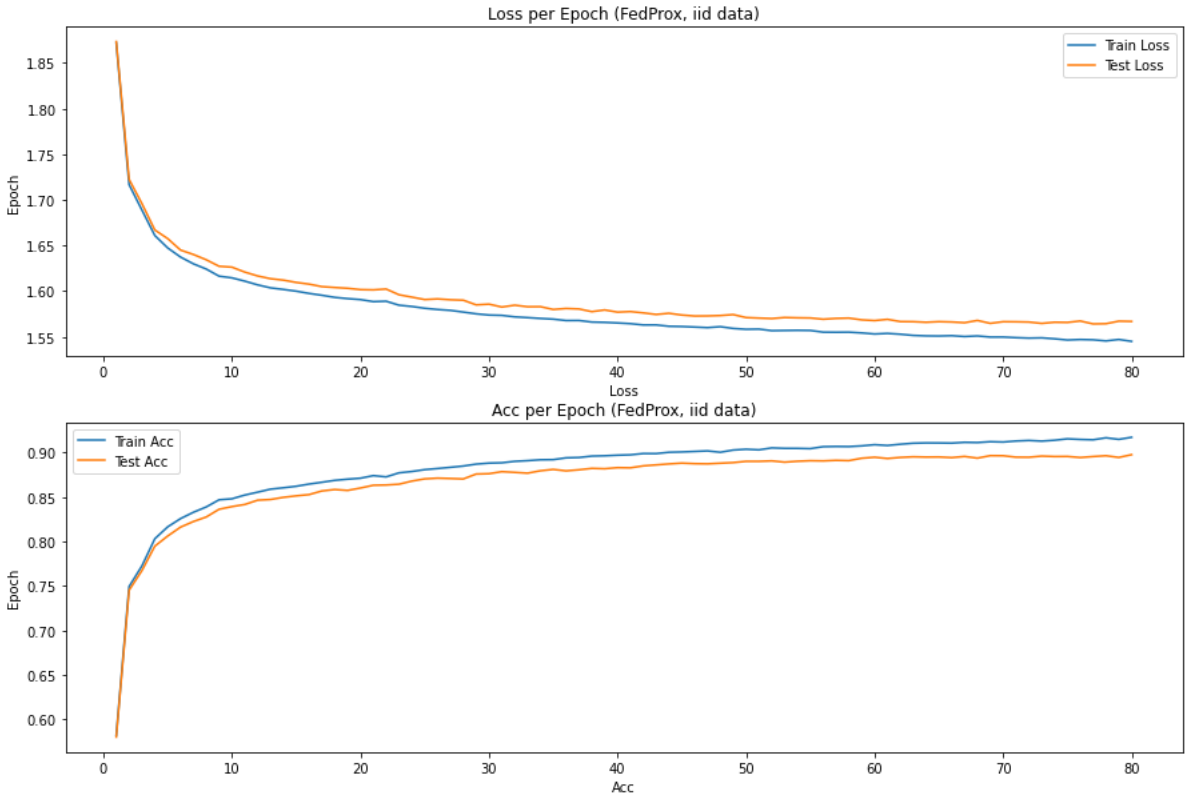

Acc per Epoch (FedAvg, iid data)
Train Acc
Test Acc
Epoch
Acc

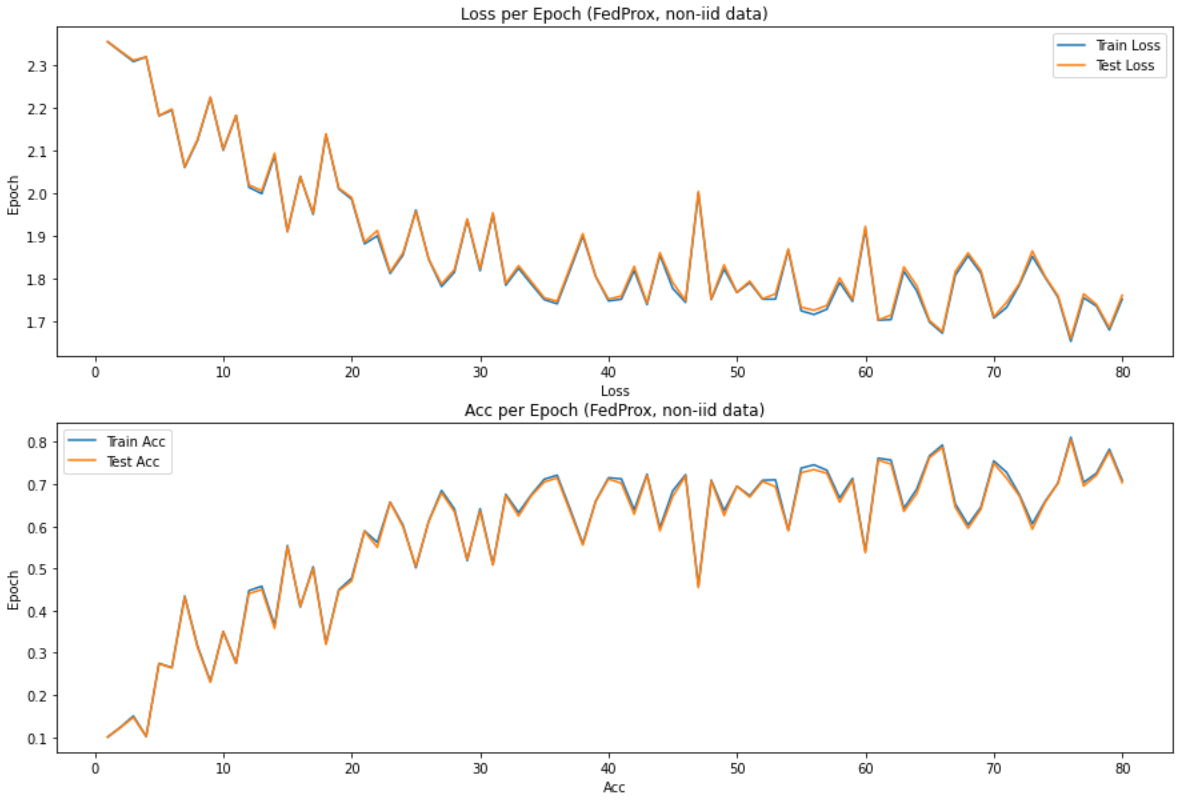

Loss per Epoch (FedAvg, non-iid data)
Train Loss
Test Loss
Epoch
Loss


Acc per Epoch (FedAvg, non-iid data)
Train Acc
Test Acc
Epoch
Acc

5.3. FedProx

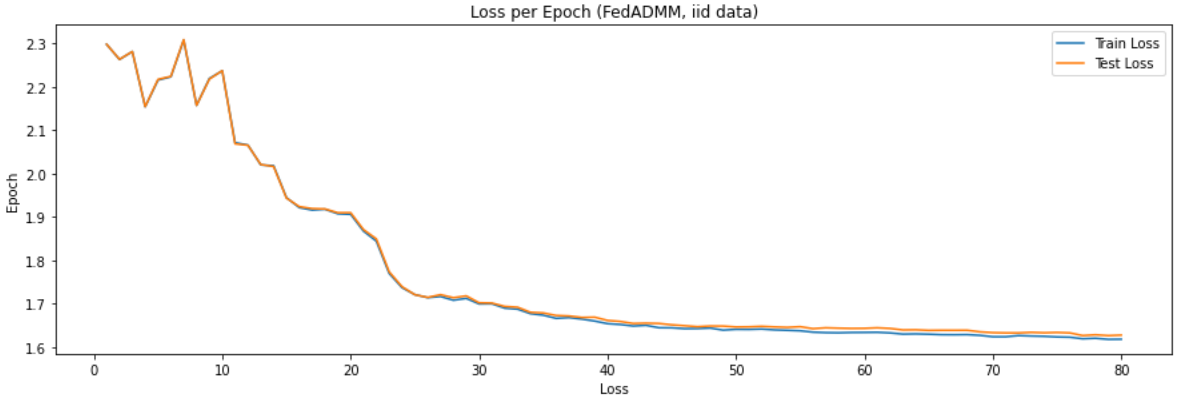
For FedProx, apart from the common hyperparameters mentioned in Section 3, we need to determine the number of rounds (E) which we choose as 80, the fraction of clients chosen at each round (k) which we set to 0.1, and the regularization parameter μ which we set to 3 after some manual tuning. The train and test loss accuracy for the global model at each round are plotted below for both iid and non-iid distributions,

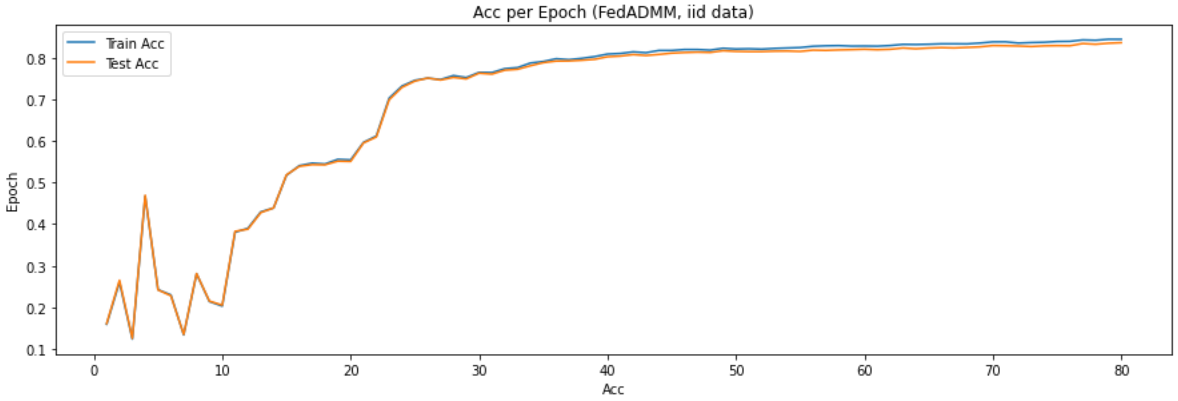


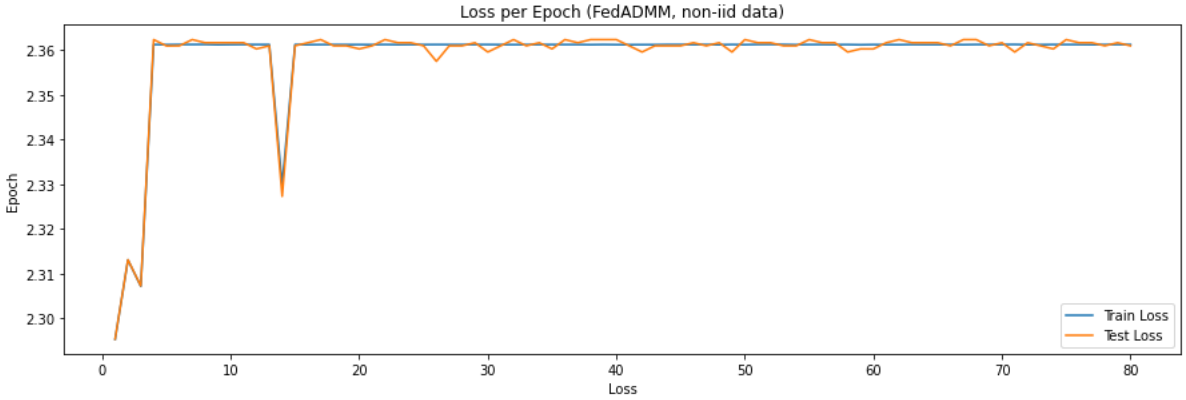


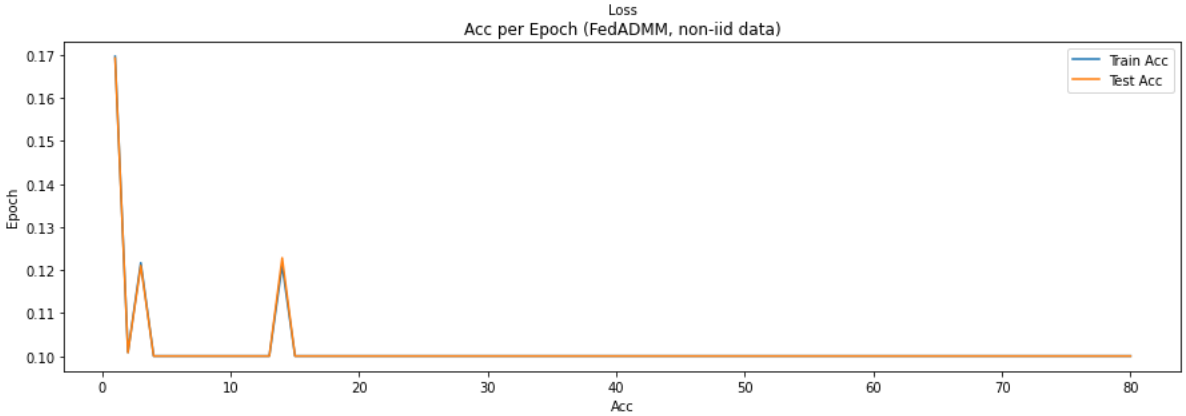
5.4. Linearized inexact ADMM-based federated learning (LIADMM)

For LIADMM, apart from the common hyperparameters mentioned in Section 3, we need to determine the number of rounds (E) which we choose as 80, and the fraction of clients chosen at each round (k) which we set to 0.1. The parameter γ should be equal to the learning rate which is 0.05 according to the algorithm, The train and test loss accuracy for the global model at each round are plotted below for both iid and non-iid distributions,


Loss per Epoch (FedADMM, iid data)
Train Loss
Test Loss
Epoch
Loss


Acc per Epoch (FedADMM, iid data)
Train Acc
Test Acc
Epoch
Acc


Loss per Epoch (FedADMM, non-iid data)
Train Loss
Test Loss
Epoch
Loss


Acc per Epoch (FedADMM, non-iid data)
Train Acc
Test Acc
Epoch
Acc

6. Discussion and Conclusion

In this project, we applied three different federated learning algorithms to the problem of image classification for both iid and non-iid dataset distributions. We can have a few observations from the results:

- The results on the train and test sets are not much different, as the global model is not directly trained on the train set, yet, it is aggregating weights from the models that have been trained on portions of the train set.
- When the datasets are iid, the three algorithms, FedAvg, FedProx, and LIADMM perform almost the same, although they cannot reach the accuracy achieved by centralized training.
- When the datasets are non-iid, the performance of both FedAvg and FedProx worsens and becomes noisy. Despite the promise of FedProx, it achieves only a slightly better (in terms of average accuracy and the amount of noise over the last epochs) performance than FedAvg. More tuning of the hyperparameters may improve the FedProx performance in the non-iid paradigm.
- The LIADMM algorithms completely fails when the datasets are non-iid. Given the novelty of this algorithm, especially for deep neural networks, the reason is still unclear. However, the good performance of this algorithm in the iid paradigm is promising, especially because the algorithm has a communication-efficient variant that could be used in real-world applications.

References

[1] McMahan, Brendan, et al. "Communication-efficient learning of deep networks from decentralized data." Artificial intelligence and statistics. PMLR, 2017.

[2] Zhou, Shenglong, and Geoffrey Ye Li. "Communication-efficient ADMM-based federated learning." *arXiv preprint arXiv:2110.15318* (2021).

[3] Li, Tian, et al. "Federated optimization in heterogeneous networks." *Proceedings of Machine Learning and Systems* 2 (2020): 429-450.

[4] Xiao, Han, Kashif Rasul, and Roland Vollgraf. "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms." *arXiv preprint arXiv:1708.07747* (2017).