Distributed Systems - Report

Ruofan Liao
University of Aberdeen
Email: u17rl21@abdn.ac.uk

***Abstract*—In this project, a CNN network and MNIST dataset were used to realize the function of handwritten digit recognition. In addition, I implemented a framework based on federated learning, which supports IID and non-IID data distribution, and realizes local training and global model aggregation by federal average algorithm (FedAvg).**

I. Setup and Running Instructions

I wrote a $README.md$ that explains in detail how to run this project, where $train.py$ and $fedrated\_learning.py$ implement centralized training and fedrated learning, respectively. You can modify the configuration and hyperparameters of the run in two files, such as "IID" or "non-IID".

II. Project Design

A. Preliminary preparation

Firstly, by using $pytorch$, I defined a CNN network and a LeNet5 network for handwriting recognition. Then, I defined a $MnistDataset$ class using the Dataset in $torch$ library to load the image data properly. Then I wrote a basic centralized training program to make sure it worked as expected.

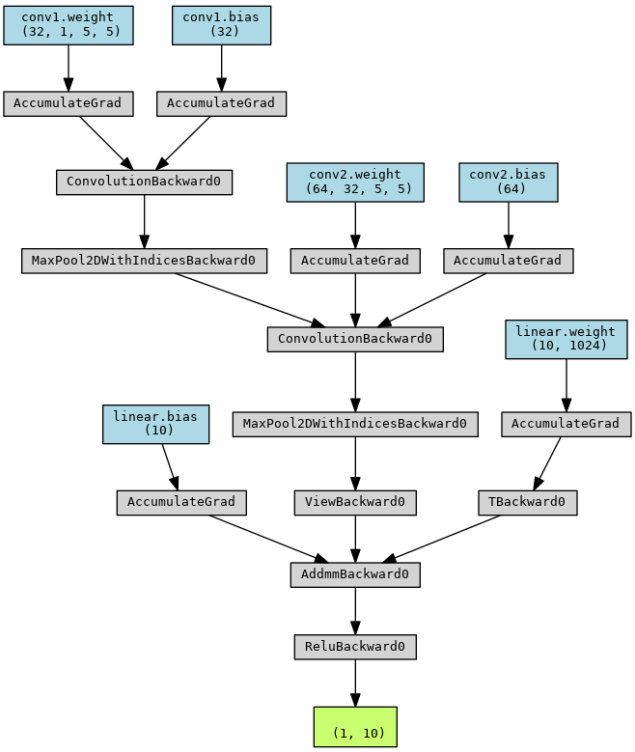

Fig. 1: Structure of the CNN network

B. Clients Simulation

By using the $defaultdict$ in $collection$ library, I split the whole dataset into multiple subsets, one for each client. The segmentation can be done in an IID (independent identically distributed) or non-IID (non-independent identically distributed) manner. The IID approach ensures the same data distribution for each client, while the non-IID approach simulates the uneven distribution of data in the real world.

Then, I initialize a separate CNN model for each client, so that each client is trained independently on its own subset of data. Through the $train\_client$ function, client_id, model, data subset, learning rate, and other relevant parameters are received, and returns the trained model parameters and performance metrics.

C. Model Aggregation

At the end of each training round, the server collects model parameters from all clients and merges these parameters using the Federated Averaging algorithm (FedAvg) to produce a global model. The server then sends the aggregated global model parameters to all clients, which use these parameters to update their local models and start the next round of training.

D. Result Evaluation and Visualization

After each training round, in the fed_avg function, the average loss and accuracy are calculated and recorded for each client. Then, draw the average loss and the accuracy in a graph by using the $matplotlib$ library. I also set a variable $threshold = 0.01$ to estimate whether the model has converged. The threshold is calculated by iterating the loss list and the difference in loss between the current epoch and the previous epoch since the second epoch. After the training ends, called the $evaluate\_model$ function, using $model.eval()$ changes the model into evaluation mode and run the test dataset to check the accuracy.

III. Training Result

All the results are trained in the same hyper-parameters: $epochs = 30$, $batch\_size = 64$, $lr = 0.001$, and $threshold = 0.01$, which means the model is considered to be converged if the difference of loss between two epochs is less than 0.01. The following figures show the result of centralized learning, IID data distribution federated learning, and non-IID data distribution federated learning with 5 clients. Other experiment results about different number of clients can be found in the $Appendix$ section.

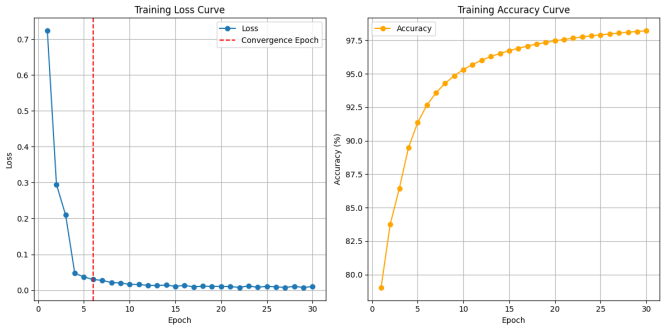

Fig. 2: Centralized Learning, 98%

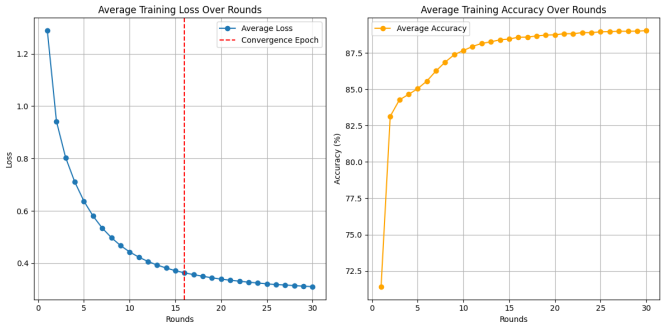

Fig. 3: IID, 5 clients, 69.53%

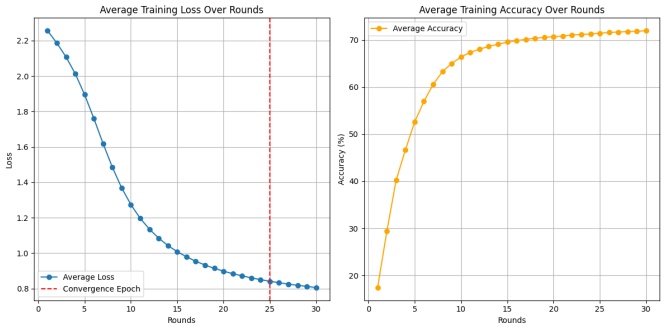

Fig. 4: Non-IID, 5 clients, 72%

According to the results, we can find that centralized learning has the best performance, and the non-IID distribution is better than IID distribution when training in same number of epochs, but it makes the model converge slower. As the number of clients increased, the model converged slower, which means more epochs of training is needed for more clients.

IV. Challenge

During the development of the project, the main challenge is about data preprocessing. After downloading the dataset from $kaggle$, the data was in the form of .idx1-ubyte and .idx3-ubyte instead of .png or .jpg. By looking up on the Internet, I found the solution online. The header of the file is parsed using $struct.unpack\_from$ to obtain the number of images, number of image rows, and number of columns. Read the image data and convert it to a NumPy array of the shape $(numImages, 1, numRows, numColumns)$, and finally save in .png form.

V. Conclusion

In this project, I have learned the concept of federated learning, implemented the CNN model training process. Finding that according to my experiments, centralized learning converges faster than federated learning, and the more number of clients the slower the model converges.

VI. Appendix

(data distribution, number of clients, accuracy on test dataset)

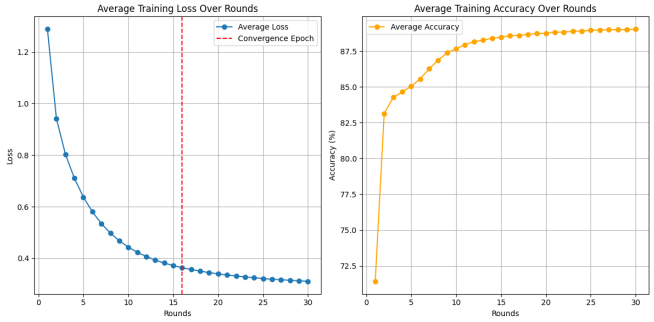

Fig. 5: IID, 5 clients, 69.53%

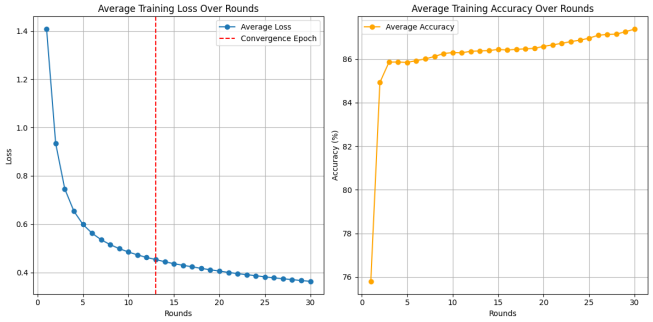

Fig. 6: IID, 10 clients, 43.69%

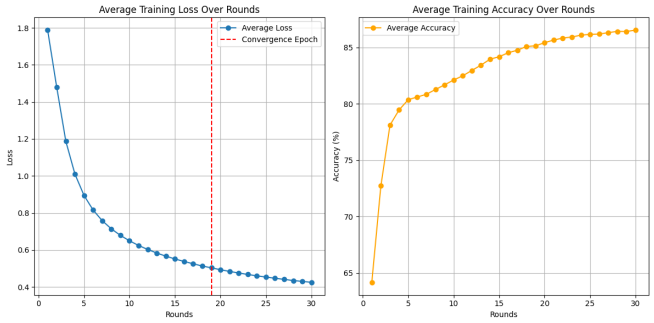

Fig. 7: IID, 15 clients, 58.05%

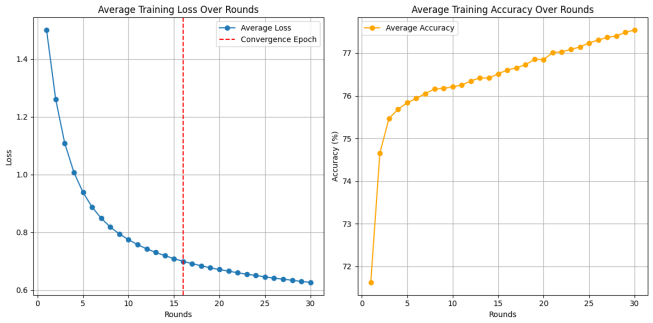

Fig. 8: IID, 20 clients, 51.667%

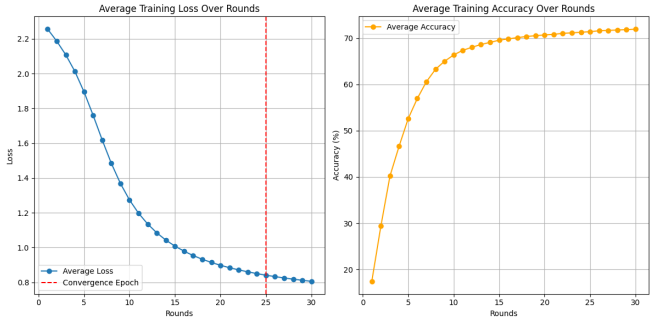

Fig. 9: Non-IID, 5 clients, 72%

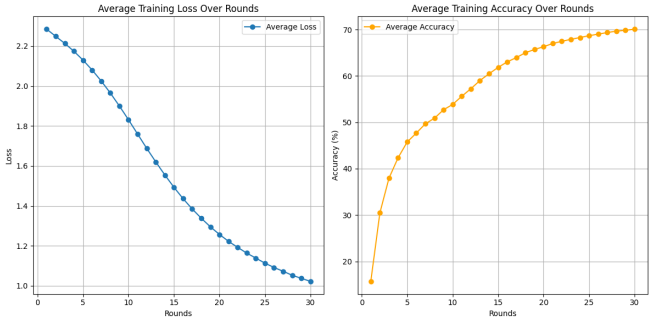

Fig. 10: Non-IID, 10 clients, 70.52%

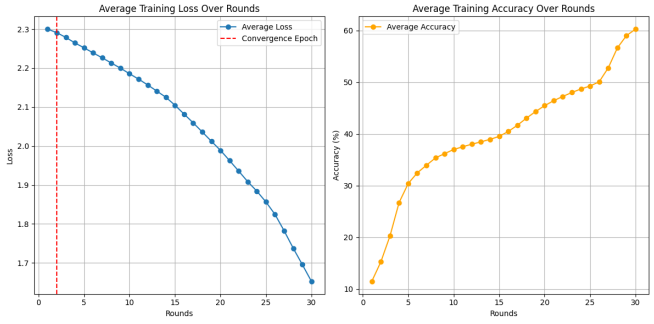

Fig. 11: Non-IID, 15 clients, 61.78%

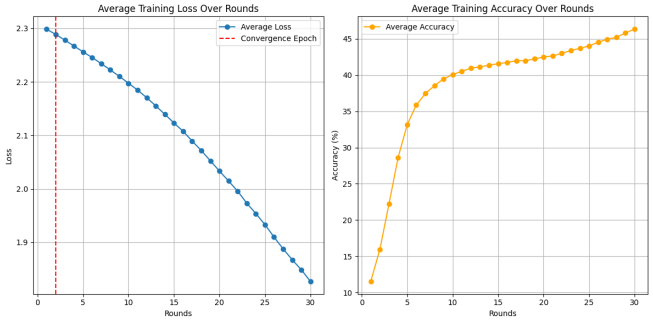

Fig. 12: Non-IID, 20 clients, 46.82%