## 1 Heterogeneous Centralized Federated Learning

The project simulates a machine learning methodology distributing data sets locally. Particularly, the system is centralized as a server is in charge of controlling the whole computation flow by exchanging information with local devices.

1. Initialization
   The server uses a multinominal logistic regression model to classify. Particularly, `model = nn.Linear(784, 10)` makes the model to label $28 \times 28$ pixel images out of 10 classifications. The initial server model $w_0$ uses a Xavier Initialization [1] with 0 initial biases for the neural network by

   ```
   nn.init.xavier_uniform_(model.weight)
   nn.init.zeros_(model.bias)
   ```

   Each client loads its own dataset, which in this assignment instance, have different sizes of rows(differnt training/test sizes). Therefore, the FL is Heterogeneous.

2. Client Selection
   Based on the flag, the server could either use all clients registered(non-subsampling) or select 2 random registered devices(subsampling) to train the server model. During each round, only selected clients receive the latest global data. The others wait for next incoming notification.

3. Configuration
   The local training method, which is either a normal gradient descent or a stochastic gradient descent with a number of training samples per epoch(Mini Batch GD in this case). The concrete training is set by the client programs before they send the handshake message to the server

4. Aggregating
   The server receives feedback updated models from the previously selected clients then normalize them into a single new global model.

5. Termination
   In this assignment, the server broadcasts to all clients to stop training after 100 global model iterations.

Between the server and one of the clients, only model parameters are exchanged. This key feature improves raw data security and prevents data set corruption as well.

## 2 GD vs. Mini-match GD performance analysis

### 2.1 Accuracy

From both Figure 1 and Figure 2, we can tell that the training loss function on iterations have the same trend with respect to iterations. However, a GD training

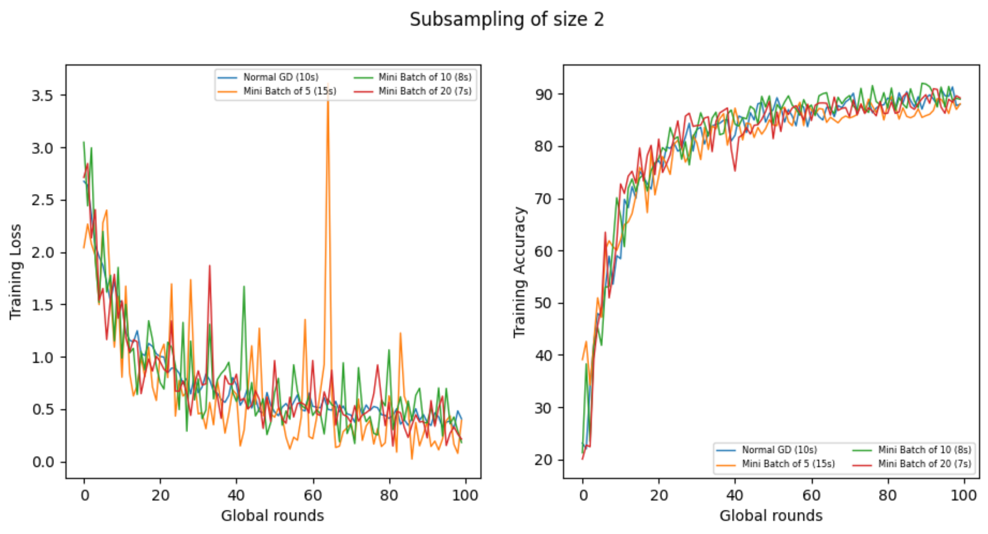

Figure 1: training methods comparison w/ subsampling

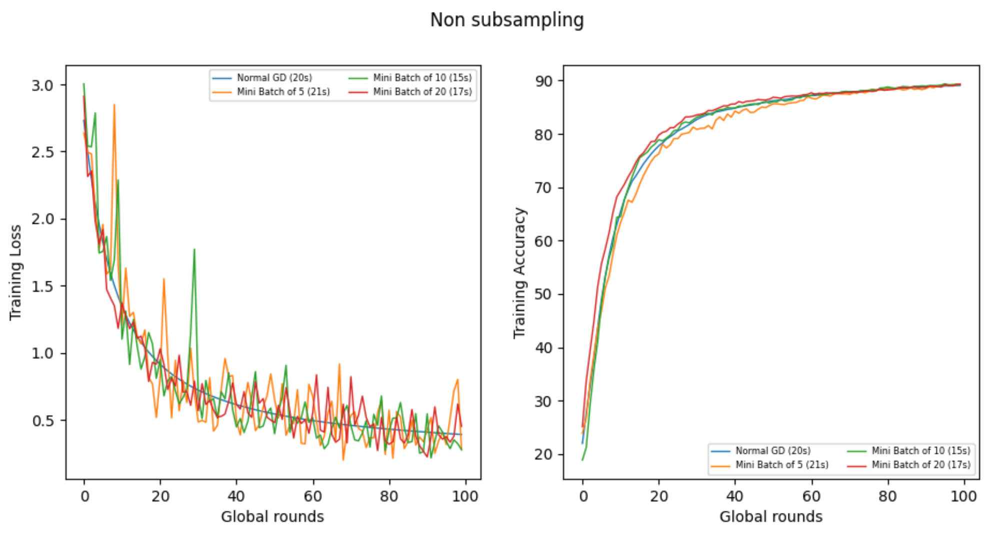

Figure 2: training methods comparison w/o subsampling

has a way smoother loss trasition compared to mini-batch GDs. As the size of batch decreases, the loss function tends to be more fluctuating around the trending regression curve. This behaviour is rational, as the batch size shrinks, the mini-batch GD is more similar as a stochastic GD(mini-batch GD size of 1). The smoothness of the scatter plot can be interpreted as that the updated model is heading to the destination in a more correct direction during the training process. This is true because GD computes the negative gradient using the whole local data while mini-batch GDs use only a small portion of the local data set.

## 2.2 Running time

The numbers in the figure's legends are the running time the server spend in completing communication rounds. As the size of batch decreases, the overall running time is expected to decrease as well since the local training process uses smaller size of data to compute. However, the FL uses sockets to communicate, as a consequence, the communication using TCP data transfer becomes an overhead. Even if a mini-batch GD of 5 entries per epoch is done fast enough, clients still need to exchange the same size of the updated model parameters. An interesting point is that the server training with mini batch 5 training used more time than the one training with normal GD. This is caused by the DataLoader object from the PyTorch.nn. The object shuffles batch units each round, which costs more resource than merely calculating gradients.

## 3 Subsampling vs. Nonsubsampling

To reduce the payload of communication, the server can choose a subset of clients to train rather than all clients. The curves in Figure 1 are "noisier" than the corresponding ones in the Figure 2 while they converge with a similar rate. There are two reasons. One is similar to using a smaller training sample size. Consider that there should be 5 batches used to train the model, instead, only 2 batches are used. We can think of using batch size of 2 instead of 5 proportionally. The other reason is the turbulence which subsampling naturally brings. Figure 3 shows how a new device

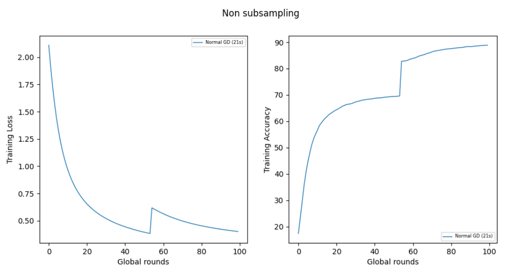

Figure 3: Client 5 joined at round 54

joining the training topology affects. In subsampling, not all clients are trained each round. Clients can be thought of disconnecting and reconnecting to the network periodically, which of course, brings more uncertainty.

## 4 Limitations of my implementation

- modified client program flags for batch sizes:
  `python COMP3221_FLClient.py client1 6001 5` for mini-batch 5.
- client port number arg doesn't matter:
  The listening/writing pipe is not statically bound to the port number. It is dynamically handled by the server program.

## References

[1] Xavier Glorot, Yoshua Bengio, *Understanding the difficulty of training deep feedforward neural networks*, DIRO, Universite de Montreal, Montreal, Quebec, Canada, 2010.