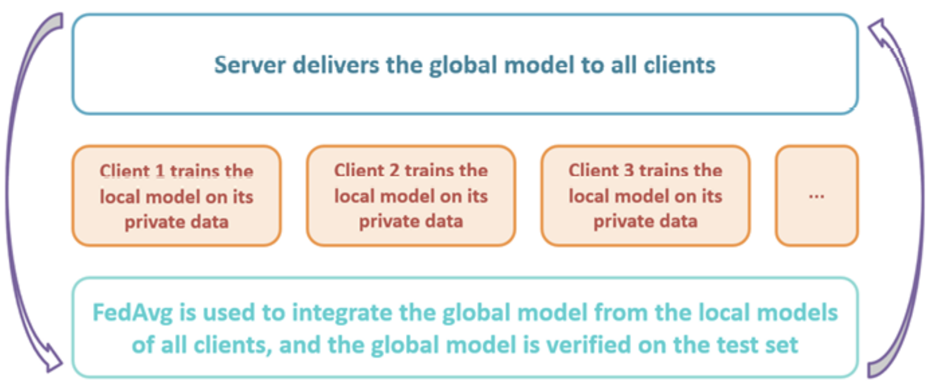
Federated Learning

Wei, 2023/5

1. Description

This is my PyTorch approach to simulate and implement the interactions between clients and the cloud server in horizontal Federated Learning mode to realize a simple MNIST classification. The details are shown as follows.

- **server**: create N threads, one thread per client
    - randomly choose M out of N clients
    - send global weight to M clients
    - receive local weight from them
    - average the weight from N clients (N-M clients will use old weight)
- **client**: create N processes, one process per client
    - receive global weight
    - train the local model on its local data
    - send local weight to the server



2. Implementation Details

2.1 FedAvg

The main algorithm used is the Federated Averaging (FedAvg) algorithm[1], which consists of alternating between a few local stochastic gradient updates at client nodes followed by a model averaging update at the server.

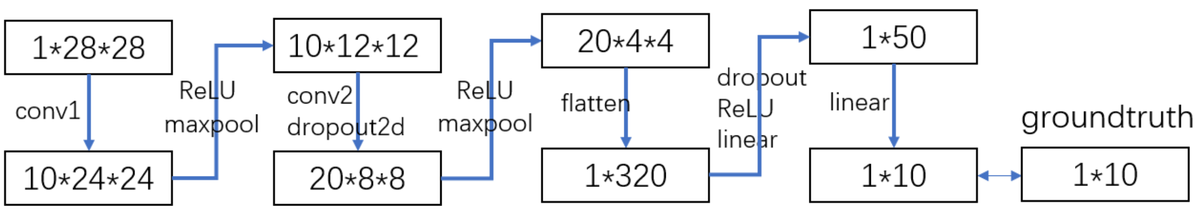
Algorithm 1 FederatedAveraging. The K clients are indexed by k; B is the local minibatch size, E is the number of local epochs, and η is the learning rate.

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

2.2 Network

The network is similar to LeNet[2], but with small difference. Cross entropy loss (realized by softmax + log + nllloss) and stochastic gradient descent with momentum are used.



2.3 Synchronization

To handle multi-threading, three condition variables are used.

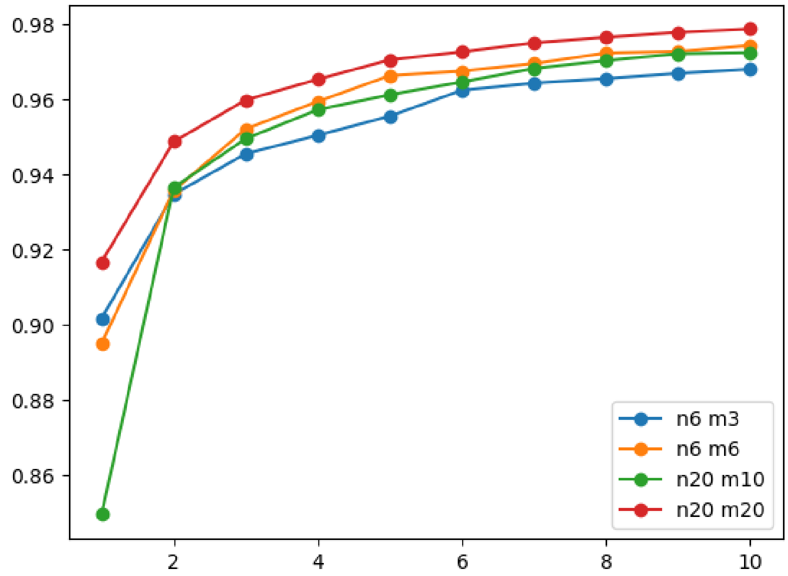
- wait until all clients are accepted
- wait until M clients have sent back the local weight
- wait until the server has averaged the weight and tested the new global network

3. Experiments

In the following two experiments, the server will stop after 10 epochs and all clients will train on their own local data in 5 epochs.

3.1 Experiment 1

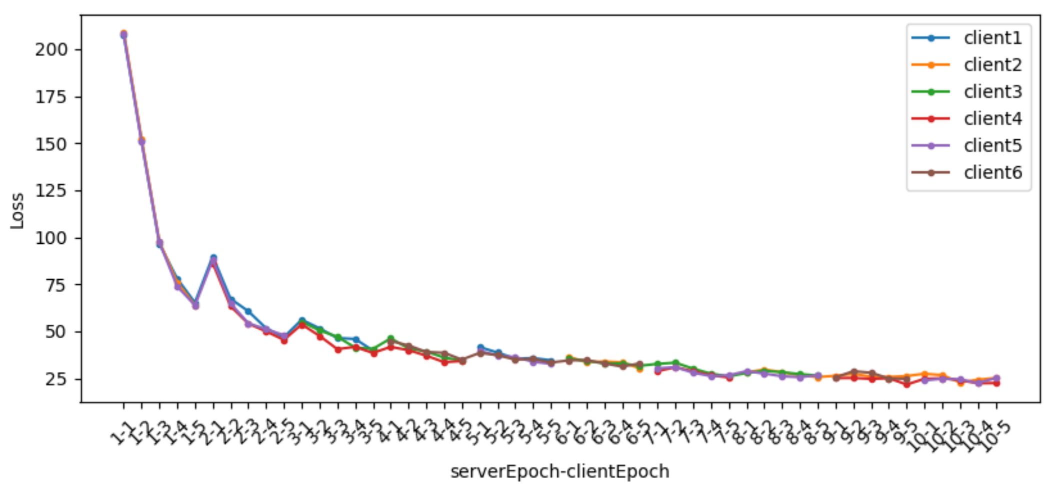
Here I test on different values of N and M, where N is the client number and M out of N will participate in the update. Here is the result of accuracy tested by the server every epoch.



- Clients' local data are IID, so the accuracy will reach a relatively high value even at the first epoch.
- When N is fixed, the larger M is, the higher the accuracy is.
- When the ratio of M and N remains the same, the larger the value of the two, the higher the accuracy.

3.2 Experiment 2

In this experiment, N is 6 and M is 3. Here is the loss of each client.



- By training other clients, a client is actually training itself.
- After averaging, the new model will have a higher loss than the previous model the client sent.

4. Reference

[1] H. Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson and Blaise Aguera y Arcas, "Communication-Efficient Learning of Deep Networks from Decentralized Data",arXiv:1602.05629,2017

[2] Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, "Gradient-based learning applied to document recognition," in *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278-2324, Nov. 1998, doi: 10.1109/5.726791.