THE UNIVERSITY OF SYDNEY

Due: 3 June 2021(Thursday Week 13) by 11:59 PM

COMP3221
Assignment 2: Federated Learning

This assignment is to be completed in groups of 2 students. You have to register your groups in Canvas. In case you cannot find a group, please contact your tutor to arrange or you will be selected randomly to the group still having a slot. The final version of your assignment should be submitted electronically via CANVAS *by 11:59PM on the Thursday of Week 13 (Hard Deadline).*

1 Learning Objectives

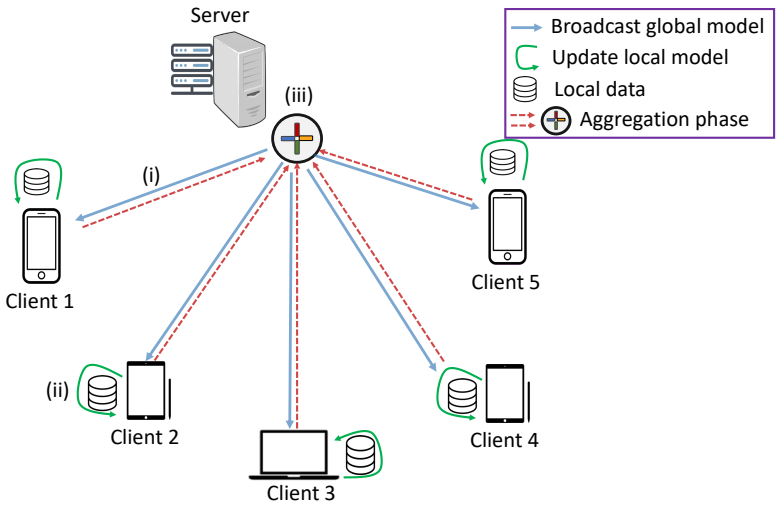

Figure 1: An example of a Federated Learning system with 5 clients.

Your task is to implement a simple Federated Learning (FL) system including five clients in total and one server for aggregation in Figs. 1. Each client has its own data used for training its local model and then contributes its local model to the server through the socket in order to build the global model. It is noted that we can simulate the FL system on a single machine by opening different terminals (one each for every client and one for server) on the same machine (use "localhost").

On completing this assignment you will gain sufficient expertise in the following skills:

- Designing distributed machine learning system
- Client-server socket programming
- Machine learning programming

2 Instructions

2.1 Federated Learning Algorithm

Algorithm 1 Federated Averaging (FedAvg), K is number of clients, E is number of local epoch, n_k is local datasize of client k, n is total datasize of K clients.

procedure SERVERUPDATE ▷ Run on server
  Generate w_0 randomly
  for t from 0 to T **do**
    Server broadcasts global model w_t to K clients
    for each client $k \in K$ in parallel **do**
      $w_{t+1}^k \leftarrow$ ClientUpdate(k,w_t)
    end for
    Server receives new local models from K clients and
    randomly selects a subset M clients in K, $M \leq K$ to aggregates new global model:
    $w_{t+1} = \sum_{k=1}^{M} \frac{n_k}{n} w_{t+1}^k$
  end for
end procedure
procedure CLIENTUPDATE(k,w_t) ▷ Run on client k
  for e from 1 to E **do**
    Client k creates new local model w_{t+1}^k from the global w_t using GD or Mini-Batch GD
  end for
  Client k sends new local model w_{t+1}^k to the server
end procedure

In this assignment, we will implement FedAvg, presented in Alg. 2.1. T is the total number of global communication rounds between clients and the server. While w_t is the global model at iteration t, w_{t+1}^k is the local model of client k at iteration $t+1$. To obtain the local model, clients can use both Gradient Descent (GD) or Mini-Batch Gradient Descent (Mini-Batch GD) as the optimization methods.

2.2 Dataset and classification model

We consider MNIST [1], a handwritten digits dataset including 70,000 samples belonging to 10 classes (from 0 to 9) for this assignment. In order to capture the heterogeneous and non-independent and identically distributed (non. i.i.d) settings in FL, the original dataset has been distributed to $K = 5$ clients where each client has different data sizes and has only 3 over 10 classes.

The dataset is located in folder named `"FLdata"` (downloaded on Canvas -> Assignment 2). Each client has 2 data files (training data and testing data) stored in 2 `json` files. For example: traing and testing data for Client 1 are `"mnist_train_client1.json"` and `"mnist_test_client1.json"`, respectively.

[1] http://yann.lecun.com/exdb/mnist/

Figure 2: MNIST - Handwritten digits dataset.

As MNIST is used for a classification problem, you are free to choose any classification methods such as multinominal logistic regression, DNN, CNN, etc... as the local model for all clients. However, all clients have to have the same kind of classification model. To simplify implementation, we recommend using the multinominal logistic regression.

2.3 Program structure

There are 2 main programs that need to be implemented: one for clients and one for the server. The server program has to be started before starting client programs.

2.3.1 Server

The server program should be named as **`COMP3221_FLServer.py`** and accepts the following command-line arguments:

```
python COMP3221_FLServer.py <Port-Server> <Sub-client>
```

For example:

```
python COMP3221_FLServer.py 6000 1
```

- **Port-Server**: is the port number of the server used for listening model packets from clients and it is fixed to 6000.
- **Sub-client**: is a flag to enable clients subsampling. (0 means $M = K$ then the server will aggregate all clients model, 1 means $M = 2$ then the server only aggregate randomly 2 clients over 5 clients).

Following Alg. 2.1, initially, the server randomly generates the global model w_0 and listens for hand-shaking messages from new coming clients. Whenever the server receives the hand-shaking message from a new client, it will add this client to a list (client's list) that contains all clients in the FL system. The hand-shaking message includes the client's data size and id.

After receiving the first hand-shaking message of one client, the server will wait for 30s to listen to new other coming clients for registration (this process only happens once when the

server starts). The server then broadcasts the global model to all registered clients and then waits for all clients to send new local models for aggregation. You are free to define the exact format of the model packets and hand-shaking messages.

After collecting all local models from all clients, the server aggregates all client's models (or subset $M = 2$ clients depends on the setting) to form a new global model and broadcasts the new global model to all registered clients. This process is one global comunication round. In this assigment, FL system will run $T = 100$ global communication rounds. For each round, server will print out the following output to the terminal:

```
Global Iteration 10:
Total Number of clients: 5
Getting local model from client 1
Getting local model from client 2
Getting local model from client 5
Getting local model from client 4
Getting local model from client 3
Aggregating new global model
Boardcasting new global model
```

If there is a new client coming for registration after the server has finished the initialization, the server will add this client to the current client's list and broadcast the global model in the next global communication round.

2.3.2 Client

The client program should be named as **`COMP3221_FLClient.py`** and accepts the following command line arguments:

```
python COMP3221_FLClient.py <Client-id> <Port-Client> <Opt-Method>
```

For example:

```
python COMP3221_FLClient.py client1 6001 1
```

- **Client-id**: is ID of a client in a federated learning network and is indexed as following client1, client2, client3, client4, and client5.
- **Port-Client** is the port number of a client receving the model packets from the server. The port number is integer indexed from 6001 and increased by one for each client. For example the port number from client 1 to client 5 are from 6001 to 6005.
- **Opt-Method**: is an optimization method to obtain local model (0 is for GD and 1 is for Mini-Batch GD).

Upon initialization, each client loads its own data, sends the hand-shaking message to the server for registration, and waits for the server to broadcast the global mode. On receiving the global model packets, the client uses this model to evaluate the local test data. It also logs the training loss and accuracy of the global model at each communication round to a file named **`client1_log.txt`** (for evaluation purpose) and prints out to the terminal, for example:

```
I am client 1
Receving new global model
Training loss: 0.01
Testing acurancy: 98%
Local training...
Sending new local model
```

After that, the client uses the global model for continuing the training process to create a new local model. The local training process can be finished in $E = 2$ local iterations using GD or Mini-Batch GD. The client then sends the new local model to the server and waits for receiving the new global model from the server. The batch-size for Mini-Batch GD can be 5, 10, or 20 samples.

2.3.3 Evaluation

To evaluate the performance of the global model across all clients at each communication round, we take the average of training loss and testing accuracy across all clients. This can be done after all clients and the server finished the training process.

3 Report:

The size of your report should be under 3 pages. Your report should briefly document your classification model, comparison of the global model performance in different settings, and meaningful discussion on the results.

The first comparison is between using GD and Mini-Batch GD methods (with different batch-size) at clients. The second comparison is to compare between subsampling clients ($M < K$) and no subsampling clients ($M = K$).

We recommend using figures to show the average training loss and accuracy of the global model at each communication round for comparison.

In addition, It should act as a reference for your markers to quickly figure out what you have and haven't completed, how you did it, and it should mention anything you think is special about your system.

4 Submission:

You are required to submit your source code and a short report to Canvas.

- Code (zip file includes all implementation, readme) **`SSID_COMP3221_FLCode.zip`**.
- Code (including all implementation in one file exported in a txt file for Plagiarism checking)
  `SSID_COMP3221_FLCode.txt`.

- Report (pdf)
  `SSID_COMP3221_FLReport.pdf`.

Please note that you must upload your submission BEFORE the deadline. The CANVAS would continue accepting submissions after the due date. Late submissions would carry penalty per day with maximum of 5 days late submission allowed.

5 Academic Honesty / Plagiarism

By uploading your submission to CANVAS you implicitly agree to abide by the University policies regarding academic honesty, and in particular that all the work is original and not plagiarised from the work of others. If you believe that part of your submission is not your work you must bring this to the attention of your tutor or lecturer immediately. See the policy slides released in Week 1 for further details.

In assessing a piece of submitted work, the School of Computer Science may reproduce it entirely, may provide a copy to another member of faculty, and/or communicate a copy of this assignment to a plagiarism checking service or in-house computer program. A copy of the assignment may be maintained by the service or the School of Computer Science for the purpose of future plagiarism checking.

6 Marking

This assignment is worth 20% of your final grade for this unit of study.

- Code: Account for 60%.
- Report: Account for 40%.