Run Instructions

Table of Contents

1. Prologue
2. Running the Blockchain Server
3. Saving the Datasets to Your Google Drive
4. Starting the Global Federated Model/Server
5. Running the Client Nodes
    a. Single Client Mode
    b. Old Multi-Client Mode
    c. New Multi-Client Mode
6. Stopping the Global Model and Clients and Collecting Results
7. Generating Graphs

Prologue

Here, we detail the instructions our team used to run our system. Please note, if you would like to run our application yourself, you are welcome to use different services, such as from AWS or Google Cloud, and adapt our codebase however you may need as certain services we used may have since changed from when we conducted our tests.

The services we used were:

- ***Replit.com***: to run our blockchain server
- ***Google Colab (GCE by extension)***: to run the global model and local nodes

In our GitHub repository, there are 9 files in the src folder. Below, we detail the three main parts of our system and which part each file falls under:

- ***Blockchain server***:
    - blockchain.py
    - blockchain-api.py
- ***Global Federated Server/Model***:
    - Server_Global_Model.ipynb
- ***Client Nodes (will need to select one based on circumstance)***:

- Local_Model_Single_Client.ipynb
- Cleaned-Latest-Local-Model-Multi-Client.ipynb
- Local-Model_Multi-Client.ipynb

The `train-test-dev-sets.zip` file has the pickled version of the MNIST dataset we used for our system.

The `Graphs_Generator.ipynb` file is not a core part of the system, rather, it is used to visualize performance results after they have been collected from running the other files in the system.

The `Baseline_Model.ipynb` file runs a standard CNN model that doesn't use federated learning and outputs its performance. It was mainly used to compare the performance of our federated models to a standard, non-federated model.

Running the Blockchain Server

To set up the blockchain server, we made a Flask repl in Python containing the two files mentioned above. In the `blockchain-api.py` file, the variable `NUM_TRAINERS` must be configured to the number of client nodes you are using for your test. After that, simply hit the run button. The console in the repl will indicate whether the run is successful and the app is listening for requests, or if there was some issue. Keep the blockchain running for the remainder of the test. To stop the blockchain server once you no longer need it running, simply hit the stop button (which should be in the same place as the run button).

Saving the Datasets to Your Google Drive

If you're using Google Colab, you'll need to save the datasets, found in the datasets folder, to your Google Colab. In both the Global Model and Client Node's code, you will need to configure the path of all `with open()` function calls to the correct location in your drive. For instance, if the variable being set by the `with open()` function is named `test_data`, you would specify the path to the test dataset in your drive.

Starting the Global Federated Model/Server

Before running it in Google Colab (or some other notebook software like Jypter), make sure you configured the following variables to what you need for your test:

`num_clients`
`rounds`
`replit_url`

Whether you choose to run each cell manually or run all cells at once, at cell 17, the global model will enter a loop where it will listen for model parameters from each client and then commit its own parameters once it receives and aggregates all client parameters. This process is repeated for as many rounds of training there are. It is here you must now switch to setting up and running the clients.

Running the Client Nodes

You have two choices regarding how you can run the clients. You can either run each client in its own Google Colab instance (this would be like running each client in its own VM), or you could simulate multiple clients in one Google Colab instance. If you choose the former, you will be using the `Local_Model_Single_Client.ipynb` file. Otherwise, you will be using either the `Cleaned-Latest-Local-Model-Multi-Client.ipynb` file (recommended new version) or the `Local-Model_Multi-Client.ipynb` (old version).

Regardless of which style you use, you must configure these variables in the code to the setting you require:

`num_clients`
`rounds`
`replit_url`

Single Client Mode

An immediate disclaimer about the single client mode is that its code hasn't been updated since our last refactor to allow the multi client code to simulate more than 8 clients. Therefore, if you are planning on running more than 8 single client instances, you will need to make changes to the single client code, similar to those made in the latest version of the multi client code. Mainly, it is regarding the division of datasets.

Whereas the new code can divide the dataset regardless of the size and divisibility, the old code can do a maximum of 8 divisions and only use `num_clients` values that can divide 8. In fact, in our original datasets folder, in addition to train, test, and validation sets, there were 8 pre-divided datasets, each named `client_train_set_#.pickle`, with # being a placeholder for some integer between 0 and 7. Unfortunately, we could

not include these partitioned train sets in our GitHub due to file size limitations, so if you don't plan on making any changes and running the original old code, you'll need to develop a script to create these partitions.

The instructions from here will detail running under the old system.

For each client, you must configure the client_id variable. Make sure that each client is assigned to a unique id, and that the id is some integer value between 0 and `num_clients` – 1 (for example, if you have 3 clients, the ids must be between 0 and 2).

After this, you can run each client one by one. Whether you run the cells individually or at once, the code for the clients will enter a loop at the very last cell when clients receive global parameters, train on their local data to generate new local parameters, send them to the blockchain, and repeat this for the number of rounds specified.

Old Multi Client Mode

The old multi client code requires you to configure the `client_id` array with a list of all possible client ids. For instance, if you had 3 clients, you must specify ids 0, 1, and 2. Alternatively, you could try using `i in range(#)` to generate all possible ids based on the number of clients.

Just like with the single client code, you must specify a `num_clients` value of no more than 8 and can divide 8.

From there, you simply run this one notebook file to simulate running multiple clients together (although the clients will be running concurrently instead of in parallel if they were each on their own machine).

The code starts looping at cell 16 for the same reason the single client code loops at the last cell – the federated learning cycles are taking place. Once this is done, running the last two cells are really optional and are a remnant of testing the code.

New Multi Client Code

To run the new multi client code, you need only configure the `num_clients` variable, and nothing else. In theory, there is no restriction on what value you could set it to, but in practice, you could start encountering issues at values higher than 64 due to how thin the dataset will be distributed across client nodes.

After that, running the code is just like running the old client code after configuration, with a loop at cell 16 and the two cells after being optional and unnecessary.

Stopping the Global Model and Clients and Collecting Results

The code for both the Global Model and Clients stop on their own. Though both the Global Model and Client's code displays data at certain parts, the main data of interest (at least in our tests) was in the Global Model Evaluation section of the Global Model's code. Cell 30 displays the Global Model's performance in various metrics at the end of each training round, cell 35 displays communication overhead stats, and cell 36 displays the final model's performance on the test set. In this section, there are other graphs and pieces of data, but the output from the aforementioned cells are the most important, especially in generating graphs later on. It is highly recommended that you copy the output from these cells and paste them into a txt file or word doc to avoid losing your results after a test run.

Generating Graphs

As mentioned before, the `Graphs_Generator.ipynb` file is used to generate graphs. Specifically, the graphs show the accuracy, loss, precision, recall, and F1 score as the number of rounds elapsed for each client configuration (along with a non-federated baseline). The way this code works is that at the beginning, there are several variables, such as `two_c` and `four_c`, that store a multi-line string of the output from cell 30 from the Global Model's notebook. Each variable corresponds to the output from a different client configuration, for instance, `two_c` stores the output from running the test with 2 clients. Running the rest of the code automatically parses these strings and creates graphs from them. If you run tests with the exact client configurations we used, you would simply paste the output string into each corresponding configuration and run the code to get the graphs. However, as is, the code is rather confusing and inflexible, so we highly recommend studying cells 5-12 to see how parsing is done for one model (the baseline model) and turn it into a general function to use for all model output strings.

Also in this file, the communication output for different client configurations is graphed. In cell 40, the following variables must be configured:

```
X
sent
received
total
```

`X` is an array which must be configured with all the client configuration used throughout tests (expressed as an integer with the number of clients used). Ideally, this list should be in ascending order. `sent` is an array that stores the bytes sent by the global model during each client configuration. These must be specified in the same order as the client configurations in `X`. The `received` and `total` arrays do the same as the `sent` array, except they store the bytes received by the global model and total bytes exchanged in that entire test, respectively. Once these variables are configured, the only thing that must be done is to run the cell and generate an output. This part of the code could also be heavily improved and we encourage the use of more refined techniques to generate graphs.