report.md

Report

- `SID: 450405907`
- `SID: 480347192`

1. Implementation details (classification model, techniques, and methodology)

- For the classification model, we used **multinomial logistic regression**.
- The `FedAvg` algorithm is used.
- The dataset used is the *MNIST*, which is a dataset consisting of handwritten digits.

The server-side code is multithreaded, the client is single-threaded. The server-side instantiates a thread for each incoming connection from the client, each of these these threads will handle the communication between that particular client with the server (example: transferring the models across the network). The client will send the handshake message which includes client ID, and the size of the training sample. Based on this information provided by the client, the server will initialise its local variables such as the total training sample size.

After the initial wait period, the client and server both run the Federated Learning algorithm. The server will first broadcast its randomly generated model to all of the clients. Client will begin to evaluate its local test data which is will be logged to a text file in the form of `client1_log.txt`. Since the server is multithreaded, synchronisation primitives (example: mutexes and barriers) were used to ensure that all the threads receive their respective local models from the clients before aggregation. Synchronisation is further used to ensure only one thread executes the aggregation function.

For the communication across the network between clients & server, socket programming and TCP protocol are used. When receiving the models from either the client or server side, the size of the model is required, therefore we follow the protocol of first sending the size of the model to the other side, so that this information is known and then sending the model across.

The system will run for *100* global communication rounds. The customisation option of specifying gradient descent or mini-batch gradient descent is also implemented. Results were collected for:

- `GD-sub`
- `GD-nosub`
- `minibatch5-nosub`
- `minibatch10-nosub`
- `minibatch20-nosub`
- `minibatch20-sub`

The script `parse_result.ipynb` is then used to parse the data into averages of loss and accuracy of global model across all clients for each communication round. The average data is then used to generate the appropriate graphs. These graphs are included in the appendix of this report.

2. GD vs. Mini-Batch GD (with different batch-size at clients)

Figure 1 (gd-vs-minibatch-accuracy) we can see that the gradient descent accuracy increases as the global communication rounds progresses, whereas the minibatch accuracy stays reletively constant. To better illustrate the differences between the various minibatch sizes, *figure 2 (only-minibatch-accuracy)* is included. In *figure 2*, we can see that the accuracy of the minibatches decreases across all the different batch sizes. By taking a look at *figure 3 (gd-vs-minibatch-loss)*, the training loss decreases for all variations. The accuracy decrease we observed for the minibatches could potentially be due to as the model reduces its training loss, it realises that it was making immature decision and the accuracy changes to better reflect reality.

3. Comparison of subsampling clients (subsample clients and non-subsample client)

In *Figure 4 (sub-vs-nonsub-accuracy)* we can see that as the global communication rounds progresses, the accuracy for both subsampling and non-subsampling increases. However, we can observe that the accuracy of non-subsampling increases smoothly, on the other hand, the accuracy of the subsampling increases in a jagged manner and fluctuates across time. In *figure 5 (sub-vs-nonsub-loss)*, we can observe that the training loss decreases for the non-subsampling but stays relatively constant for the subsampling. This indicates that as the global communication rounds progresses, the subsampling aggregation fails to create an accurate model with lower training loss, therefore in *figure 4* the accuracy is fluctuating for the subsampling model, whereas the non-subsampling model is able to produce a more accurate model as its training loss decreases across time.

4. Development process

Problems:

One of the challenges we encountered during this implementation process was the synchronisation of the multithreading on the server side. We needed to synchronise the threads both in a serial and parallel manner according to our situational needs. In order to create the desired flow control, various synchronisation primitives were used such as mutexes and barriers. Mutexes were used to protect critical areas and avoid race conditions. Barriers were used to coordinate all the server threads to ensure that the aggregations are done synchronously and by one server thread. Synchronisation primitives were also used to block the server threads until the 30 seconds mark has been reached.

Another issue was the manner in which we serialise the models into binary for sending over the network. To resolve this issue, the `pickle` library is used to serialise the models to prepare it for transfer over the TCP protocol.

We tried to support the feature in which a new client can register after the server has completed initilization, however we were unable to do so. However, other than this minor hiccup, everything else has been implemented as per the assignment specification sheet.

appendix.md

Appendix

Figure 1: gd-vs-minibatch-accuracy:

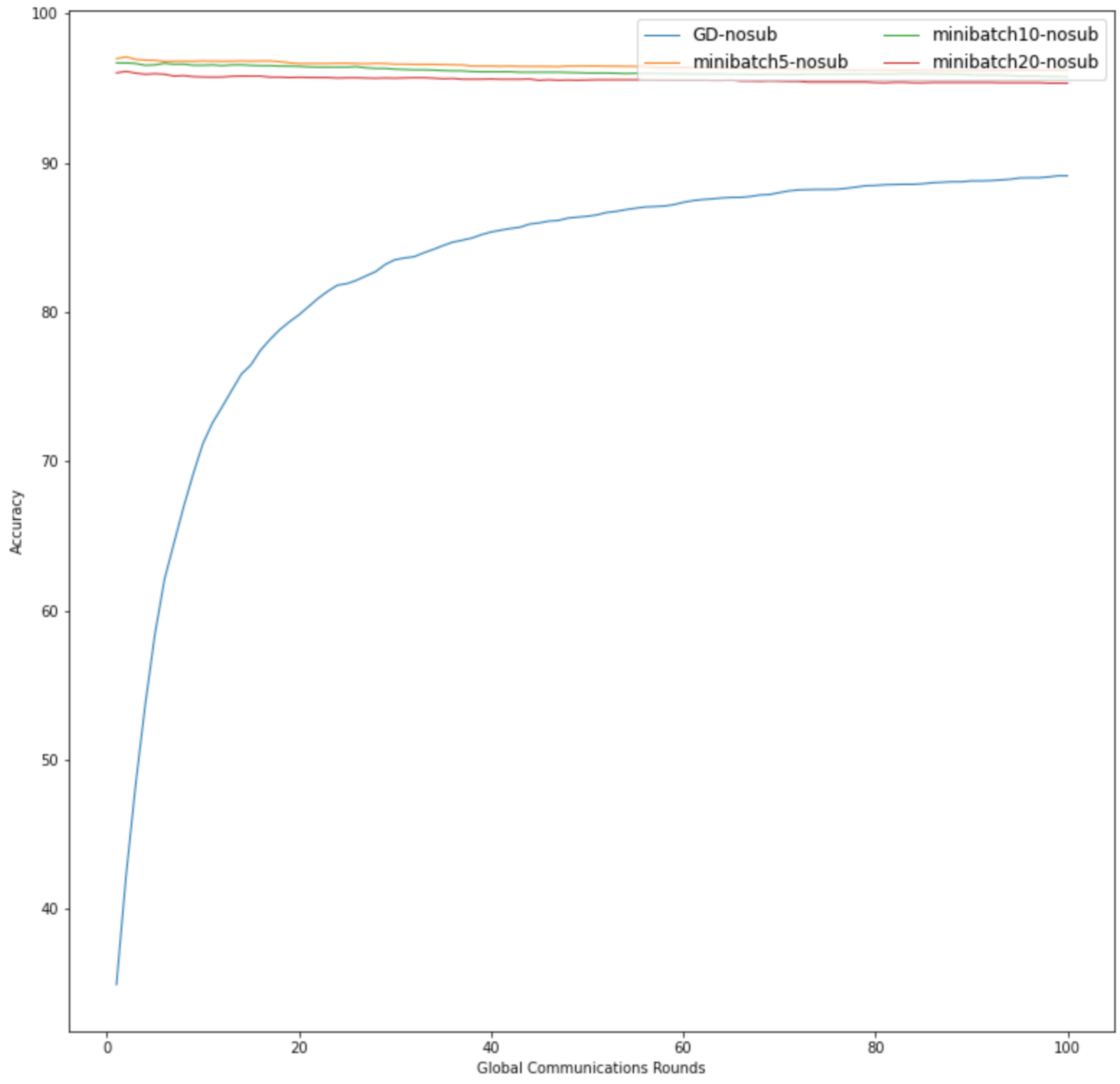


Figure 2: only-minibatch-accuracy:

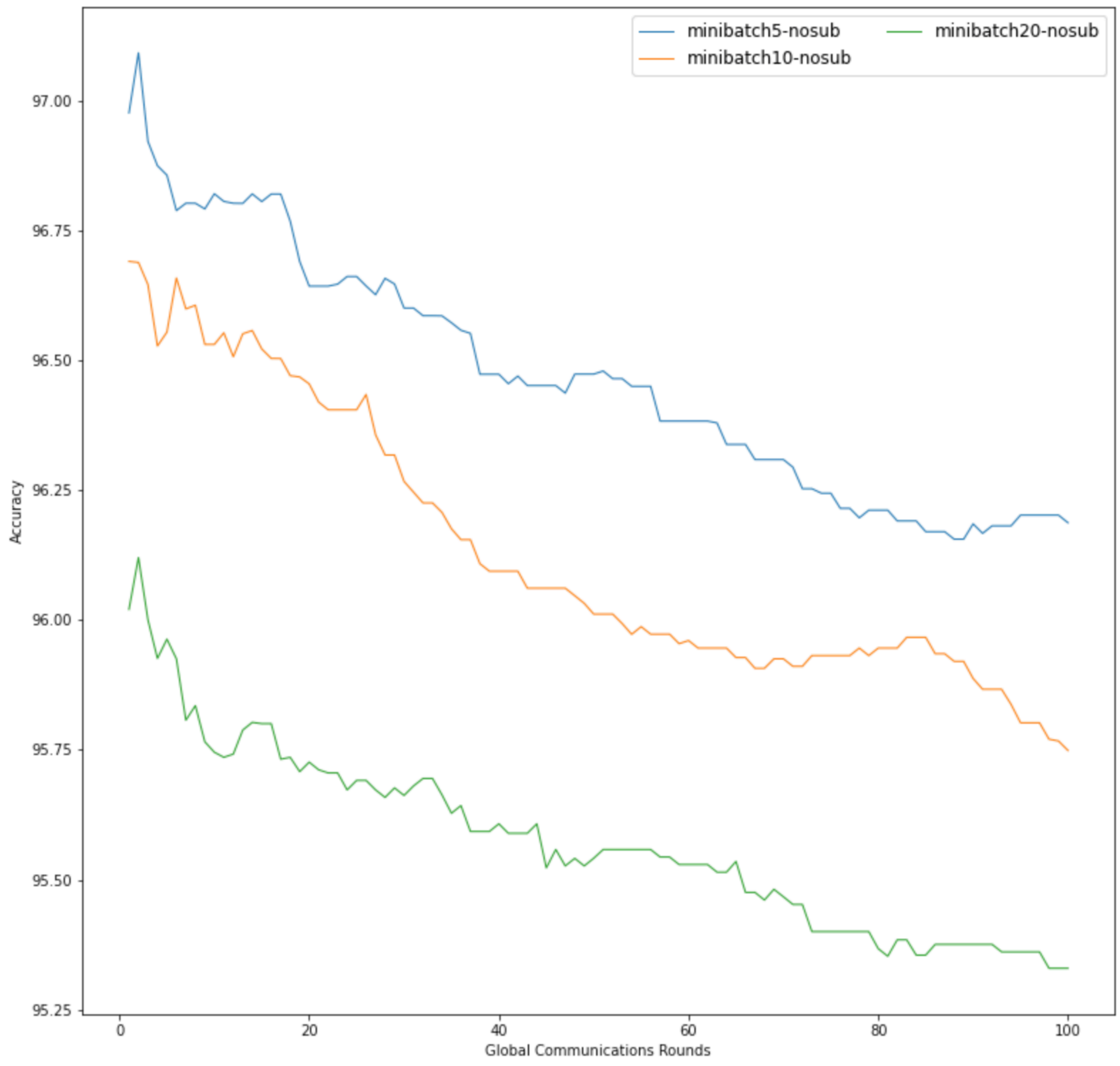


Figure 3: gd-vs-minibatch-loss:

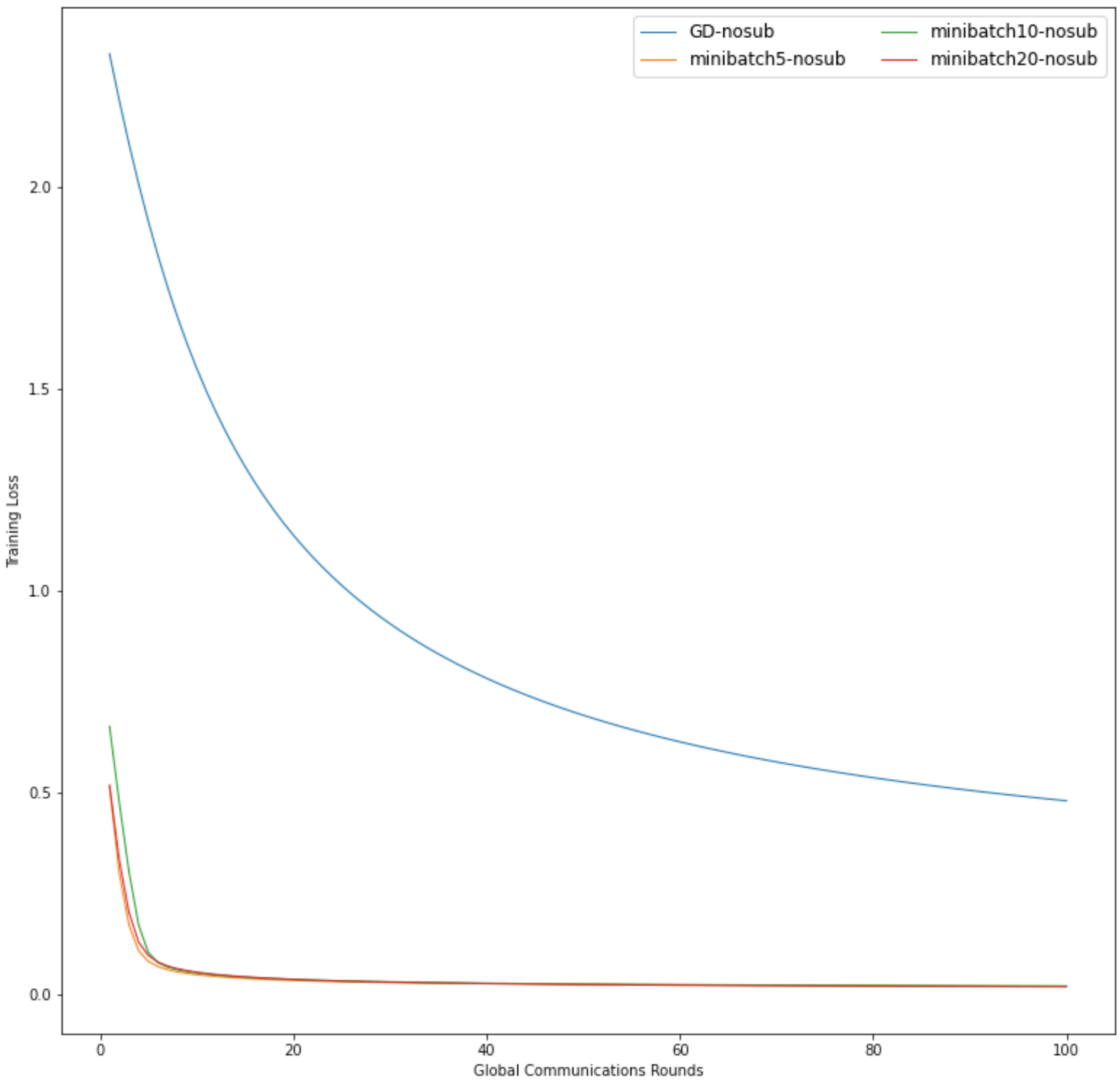


Figure 4: sub-vs-nonsub-accuracy:

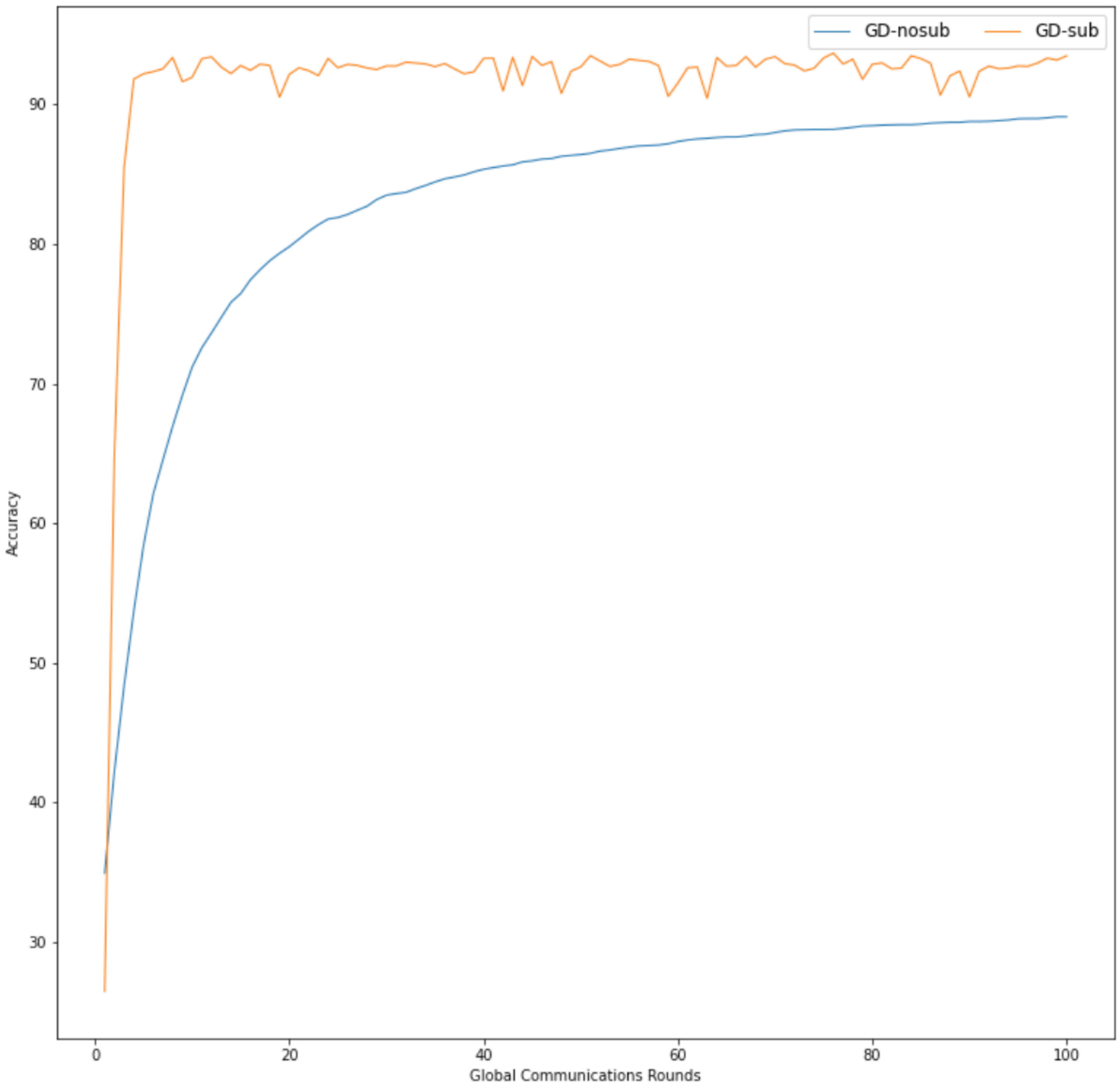
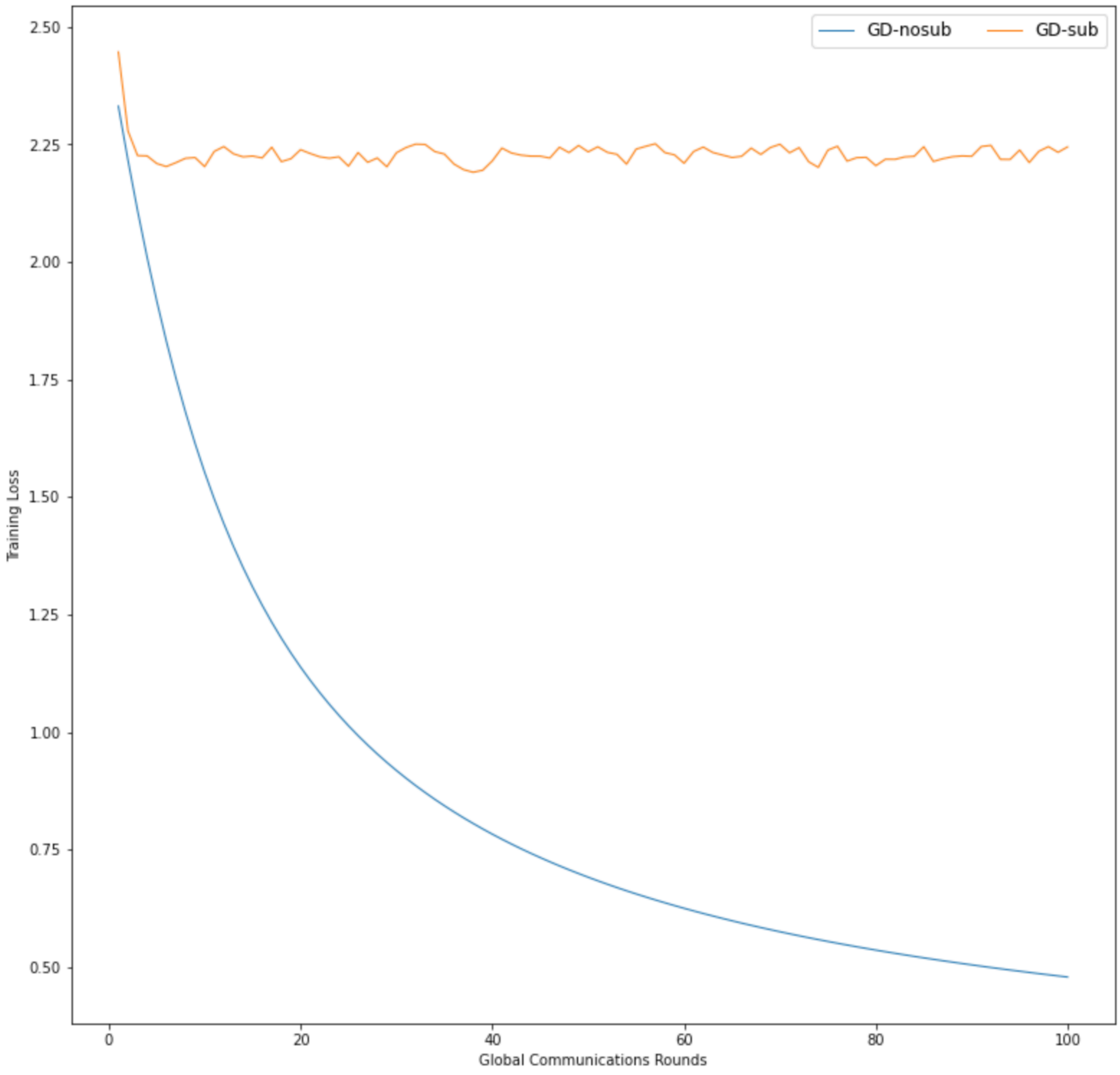


Figure 5: sub-vs-nonsub-loss:


GD-nosub
GD-sub
Training Loss
Global Communications Rounds