109006247 Report

Code Explanation

userbase.py

- set_parameters: using state_dict() to get the dictionary of parameters and their value, then calculate each parameter and store in global_para_dict before load value of each parameters into user's model using load_state_dict().

serverbase.py

- select_users: import random and use random.sample to randomly picked user with desired number [num_users] only if [num_users] is less or equal to the number of existing users.
- aggregate_parameters: starting by summing up all the number of samples for weight calculation. Then clone the parameters of the first user times by weight before adding up the rest user's parameters times by each user's weight. All done by state_dict() and load the final result by load_state_dict() similar to set_parameter in userbase.py.

The Cause of Problem

There are two main part of this federated learning process where I can implement including userbase.py, where I can determine how I use global parameters to learn through beta, and serverbase.py, where I can determine how to select user for training and do weighted sum for parameters based on the number of train samples.

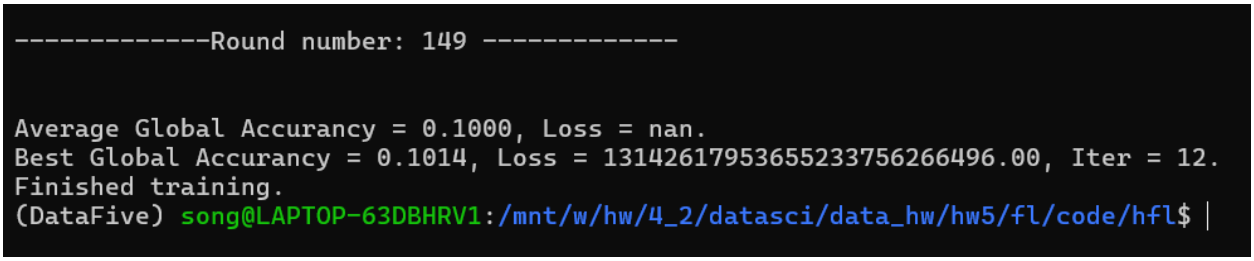
At first, I try to print out everything to see what the characteristics of the data I am dealing with. The users are the usual list, so it quite easy to randomly sample the user and finish select_user function in serverbase.py. However, parameters of the model consist of several pytorch and numpy elements which are quite hard to deal with. I have 2 ideas of how to finish another 2 functions which are either using dictionary or list. I decided to use state_dict to get the list of parameters and run through it which is easier to make change to improve the training process. I have considered changing to using list to implement the function in hope to speed up the training process, but the speed change isn't significant enough.

The result for the best accuracy of each training session to be around 0.1 expected to be from high number of users compete to train the model, resulting in high convergence rate and high learning rate lead to low in accuracy. My thought on improving the performance without changing the command provided by instruction in run.sh is to change the value of beta inside set_parameters in userbase.py which determines how much portions of user's parameters will be shifted to the values of global parameters, and another idea is change the strategy of aggregating the parameters.

However, since the average training time for 1 command using my laptop is around 8 hours, I could try those ideas before the deadline is met.

Final Accuracy Output

Figure 1. Final Accuracy Output of training with dataset CIFAR10-alpha100.0-ratio1.0-users10 and number of selected users of 10



What I have learned

Figure 2. Accuracy Output of training with dataset CIFAR10-alpha50.0-ratio1.0-users10 and number of selected users of 10

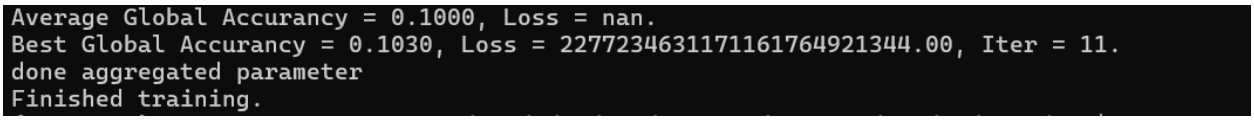
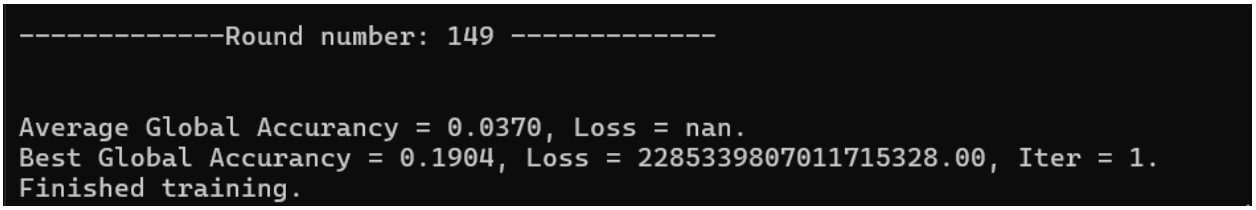


Figure 3. Accuracy Output of training with dataset CIFAR10-alpha0.1-ratio1.0-users10 and number of selected users of 10



The distribution of each user's data under scenario where alpha is equal to 50.0 is that the data consist of every class ("c") in almost equally distributed number of samples. On the other hand, the distribution of user's data with alpha equal to 0.1 is that the data only contains few classes and with different number of samples in each class. The result accuracy of alpha0.1 is almost two times that of alpha50.0. However, the convergence rate of alpha0.1 is greatly higher than alpha50.0. This alpha is used for np.repeat which generates the number of repeated data while it randomly pick the data for making new dataset to train. This has shown me the significance of having diverse data for training instead of one with identical values.

Figure 4. Accuracy Output of training with dataset CIFAR10-alpha50.0-ratio1.0-users10 and number of selected users of 2

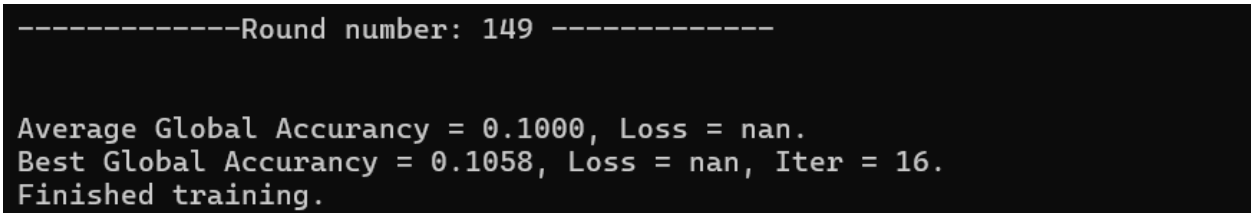
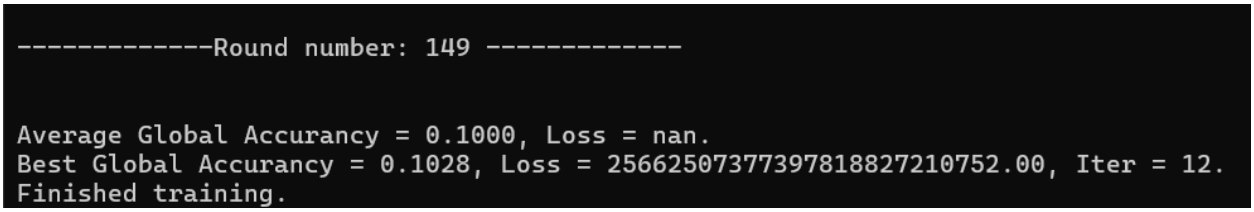


Figure 5. Accuracy Output of training with dataset CIFAR10-alpha50.0-ratio1.0-users10 and number of selected users of 10



The accuracy of training with only 2 selected users is a little bit higher than having all users training in each iteration, which strange considering the fact that more user resulting in more communication and better accuracy. I found that this is because the value of beta in set_parameters in userbase.py is equivalent to 1 which means that all users will share the global parameters without diversity. This makes the strategy of selecting only 2 users out of 10 resulting in more diversity and better accuracy. For the convergence rate of these two models is as expected that training with all users have higher convergence rate which is the result of higher communication for optimal solutions.