Federated Learning

1. Code explanation

   Federated learning allows the central server to learn from decentralized data without direct access to individual data samples, ensuring data privacy and security. It enables collaborative learning accross multiple devices while perserving the local ownership and control over data.

   ▼ serverbase.py

   In this module, we define a parent class **Server** that serves as a parent class inherited by FedAvg class in server.avg module.

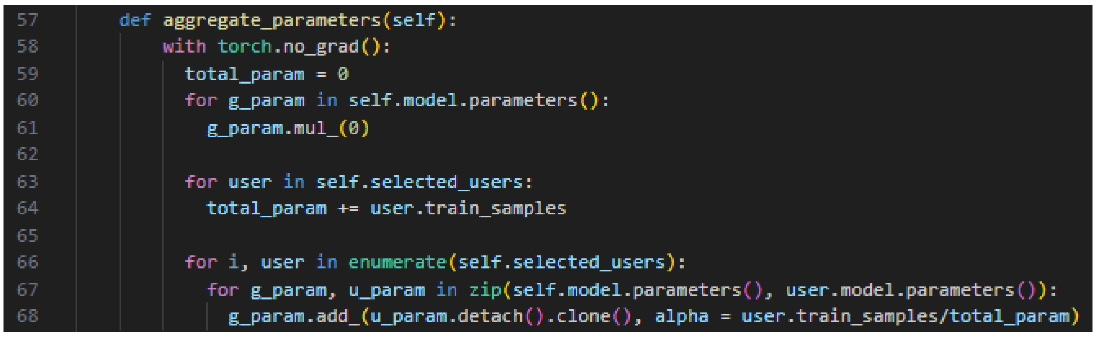
   - aggregate_parameters

   In this function, since we don’t need gradient information, I specifically add line 58 to improve memory efficiency, since pytorh will be converted to dynamic computational graph in the backend.

   Then, I initialized every value in the global model parameters, as denoted as g_param in the code snippet, to 0.

   Afterwards, I calculated the total number of samples, which will be used as the denominator in the following averaging process.

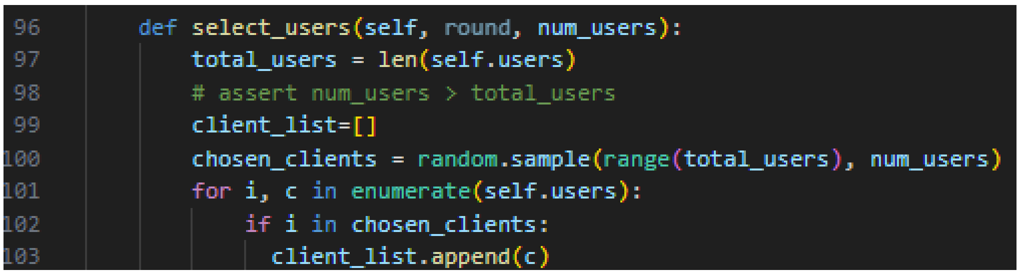
   Last, by zipping the global model with local model, we got two parameters from global model and user model. Since the global model parameters are initialized in line 61, I simply sum up the weighted average local model parameter as the global weights.



```
57      def aggregate_parameters(self):
58          with torch.no_grad():
59            total_param = 0
60            for g_param in self.model.parameters():
61              g_param.mul_(0)
62
63            for user in self.selected_users:
64              total_param += user.train_samples
65
66            for i, user in enumerate(self.selected_users):
67              for g_param, u_param in zip(self.model.parameters(), user.model.parameters()):
68                g_param.add_(u_param.detach().clone(), alpha = user.train_samples/total_param)
```

   - select_users

   In this function, the central server has to randomly select users to train on the parameters. So what we have to do is to randomly select n users from all the participating devices, returning back a list of selected user objects. So I simply initialize a client list which will store the chosen user object and chose the user by the sample function from python random package.

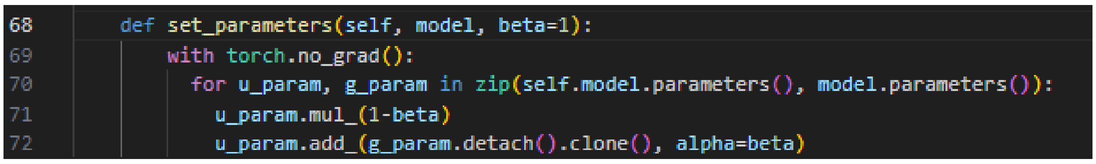


```
    def select_users(self, round, num_users):
        total_users = len(self.users)
        # assert num_users > total_users
        client_list=[]
        chosen_clients = random.sample(range(total_users), num_users)
        for i, c in enumerate(self.users):
            if i in chosen_clients:
              client_list.append(c)
```

▼ userbase.py

- set_parameter

This function is triggered when the global model finished updating by the weights from every local user parameters and redistributed the new parameters to the users. So what we have to do is to simply assign the global parameters to local parameter and retrain on the local data again. Similar to the aggregate_parameter function I mentioned earlier, I avoid memory overhead by introducing code on line 69, then zip user model and global model, finally assigning weight based on beta paramters.



```
68      def set_parameters(self, model, beta=1):
69          with torch.no_grad():
70            for u_param, g_param in zip(self.model.parameters(), model.parameters()):
71              u_param.mul_(1-beta)
72              u_param.add_(g_param.detach().clone(), alpha=beta)
```

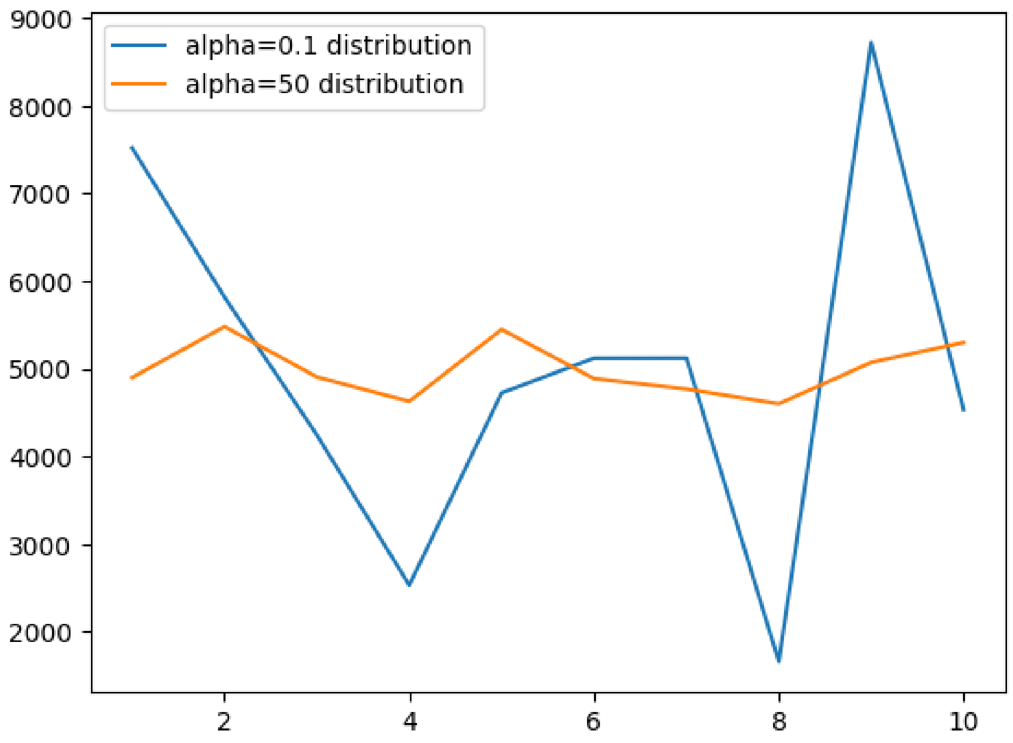
2. Problem Exploration

▼ Data Distribution

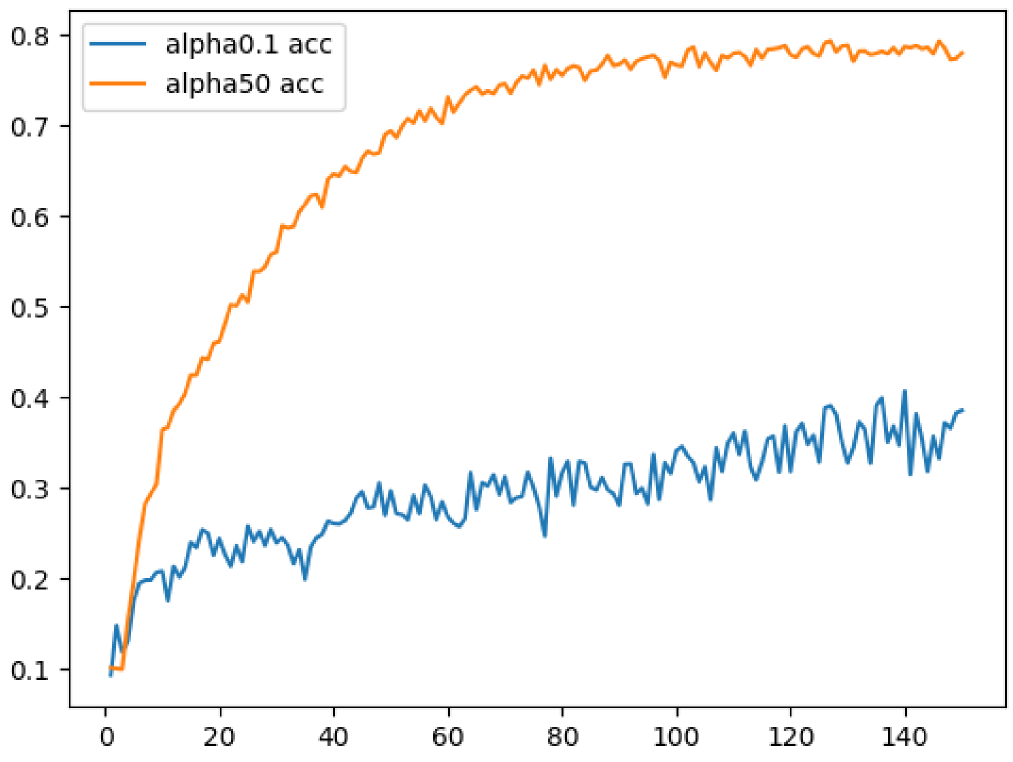
Before digging into the experiment, I surveyed on Dirichlet distribution a bit, for one, it would make writing my report a lot easier, for another, this distribution is often used in federated learning for modeling the distribution of the data across participating devices. Why is Dirichlet distribution popular for federated learning data? This is because the distribution is capable of providing a probabilistic model for describing the relative weights of data sampels among the edge devices.

In federated learning, we set the parameter α to 0.1, 50 and 100 respectively. The α value indicates a difference in the level of spread of the data among the clients. This α parameter controls the shape of the distribution, so if we set α to 0.1, it means that the distribution is more spread out, has data distribution tends to be more heterogeneity. On the other hand, if we set α to 50, the distribution among local data would be more approximated to uniform distribution. And in this experiment, I would like to examine would smaller value of α, larger heterogenity, leads to better performance or it is the opposite?

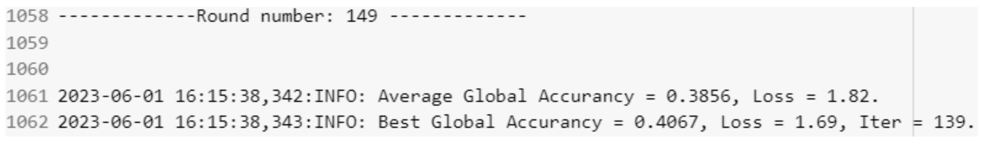
▼ $\alpha = 0.1$ vs. $\alpha = 50$ - data distribution plot



▼ $\alpha = 0.1$ vs. $\alpha = 50$ - acc

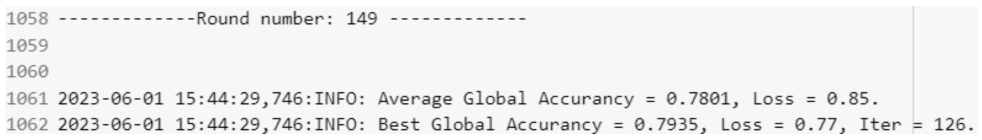


- $\alpha = 0.1$



```
------------Round number: 149 ------------


2023-06-01 16:15:38,342:INFO: Average Global Accurancy = 0.3856, Loss = 1.82.
2023-06-01 16:15:38,343:INFO: Best Global Accurancy = 0.4067, Loss = 1.69, Iter = 139.
```

- $\alpha = 50$



```
------------Round number: 149 ------------


2023-06-01 15:44:29,746:INFO: Average Global Accurancy = 0.7801, Loss = 0.85.
2023-06-01 15:44:29,746:INFO: Best Global Accurancy = 0.7935, Loss = 0.77, Iter = 126.
```

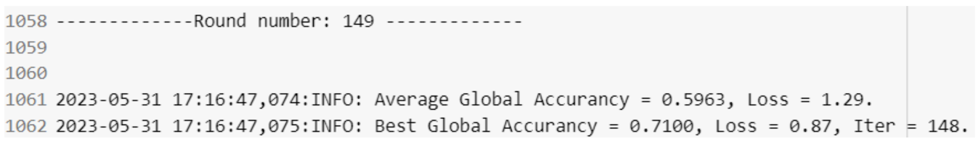
As we can observe from the experiments, $\alpha = 0.1$, which represents a higher degree of heterogeneity or say diversity in the data distribution, with more varied proportions of data samples across the participating devices performs worse comparing to when the . On the other hand, $\alpha = 50$ represents a mroe homogeneous or approximated-uniform data distribution, so this federated training task benefits from a more consistent and standardized training process, by setting a larger value of α, which indicates the users have more balanced proportions of data samples, with less variation among the all the local devices.

But why can consistency among user data can potentially lead to better performance? First, when the data distribution is more consistent across devices, it reduces the potential for bias in the training process. Next, consistency in data distribution can improve the communication efficiency between the device and the central server. Devices with similar data distributions may generate more compatible updates, requiring fewer communication rounds to read a concensus. This not only reduces the overall communication overhead, but speeds up the federated learning process.

▼ Number of users in a round

▼ Two users

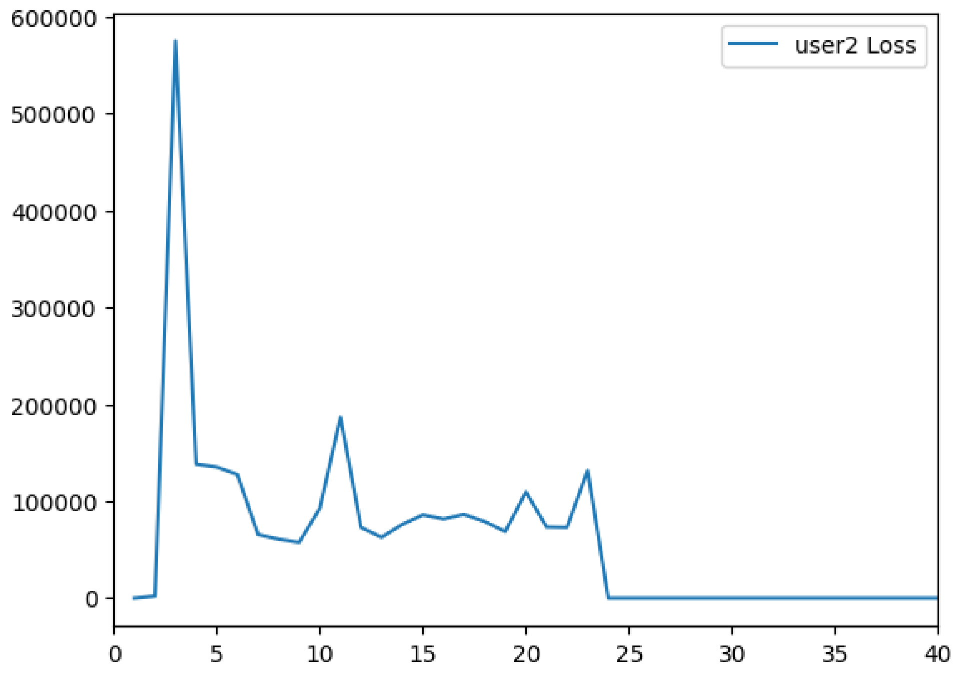
- Acc



```
-------------Round number: 149 -------------


2023-05-31 17:16:47,074:INFO: Average Global Accurancy = 0.5963, Loss = 1.29.
2023-05-31 17:16:47,075:INFO: Best Global Accurancy = 0.7100, Loss = 0.87, Iter = 148.
```

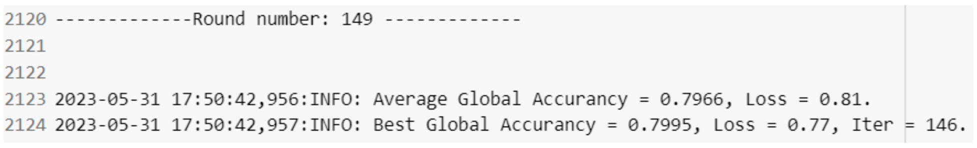
- Loss

The contral server converges after about 20 training epochs.



▼ Ten users

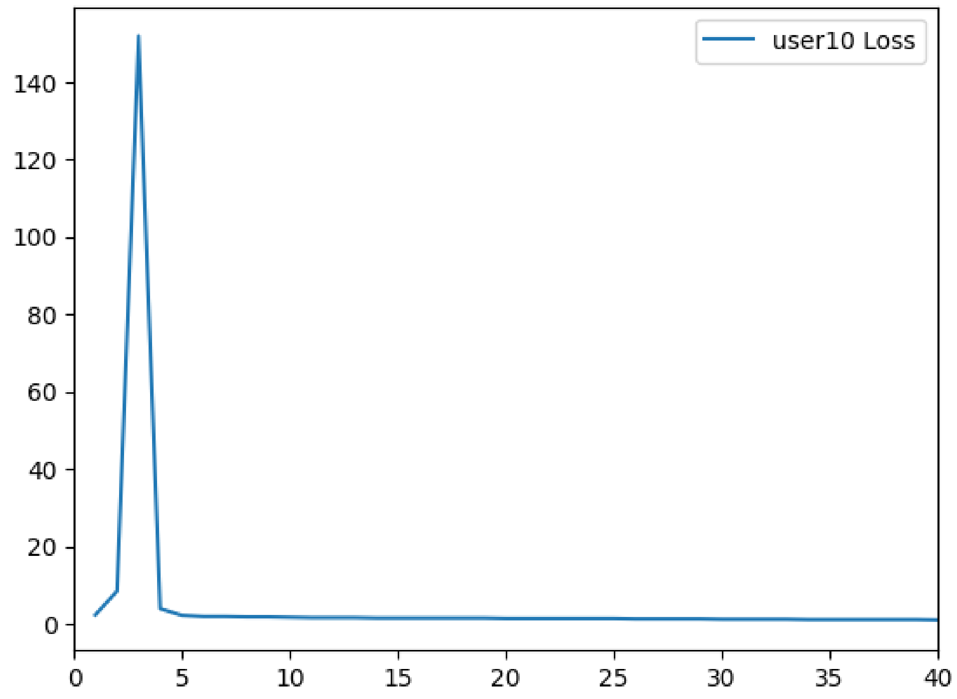
- Acc



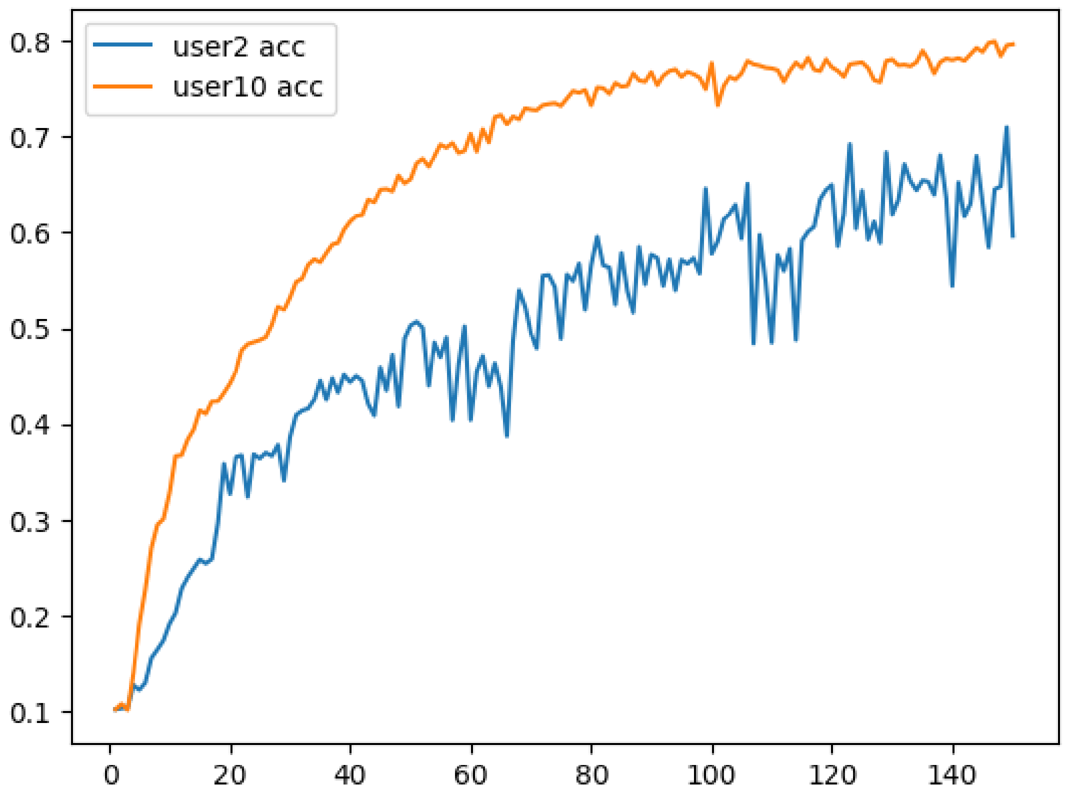
```
------------Round number: 149 ------------


2023-05-31 17:50:42,956:INFO: Average Global Accurancy = 0.7966, Loss = 0.81.
2023-05-31 17:50:42,957:INFO: Best Global Accurancy = 0.7995, Loss = 0.77, Iter = 146.
```

- Loss

  The central server converges after a few epochs.



- Global model acc



▼ Comments

It is worth noticing that with more participating clients, the central server has a tendency to converge faster and has better performance.

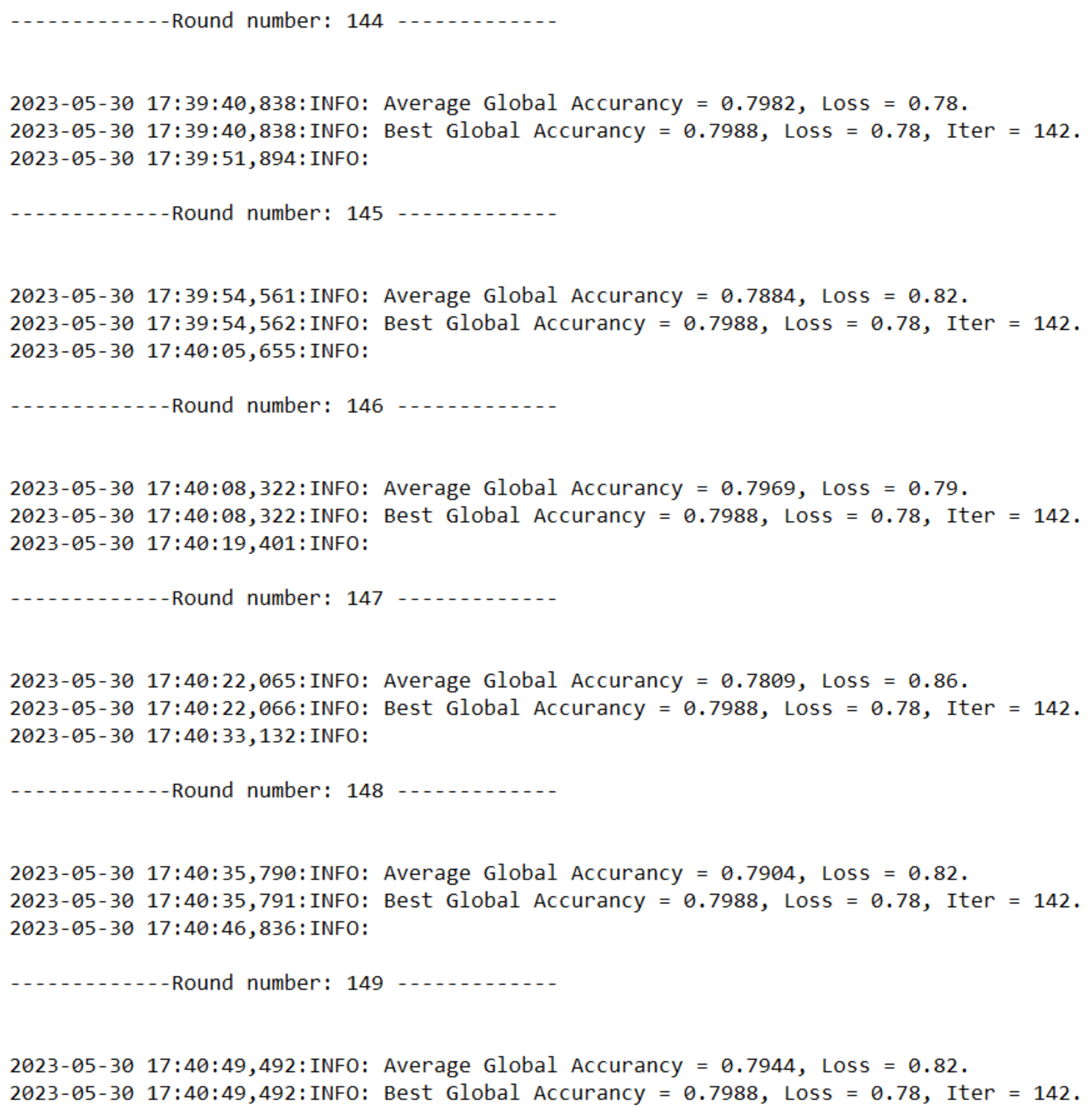
In my opinion, with more users, each user can access to different subsets of data, resulting in the central server to aggregate with parameters trained on diverse data source. This diversity can contribute to a more comprehensive representation of the overall data distribution, helping the model generalize better. Hence, by aggregating updates from more users, the central server can benefit from a wider range of perspectives, leading to faster convergence and accuracy.

Since each edge device trains the model using its local data and can potentially capture uique patterns, local features. Aggregating these insights from multiple devices allows the central server to combine the strengths from various sources. The collective intelligence gathered from diverse users can lead to better overall performance as well.

Last but not least, the aggregation process helps metigate the bias that may exist in individual devices' local models. Each participating device contributes its local updates, allowing the central server to balance out biases and learn a more representative model. Hence, with more users chosen, the central server has access to a larger pool of updates, which helps reducing device-specific biases and result in a more accurate model.

3. Accuracy screenshot

```
python main.py --dataset CIFAR10-alpha100.0-ratio1.0-users10 --algorithm FedAvg --
num_glob_iters 150 --local_epochs 10 --num_users 10 --learning_rate 0.1 --model re
snet18 --device cuda --result_path final
```



```
-------------Round number: 144 -------------


2023-05-30 17:39:40,838:INFO: Average Global Accurancy = 0.7982, Loss = 0.78.
2023-05-30 17:39:40,838:INFO: Best Global Accurancy = 0.7988, Loss = 0.78, Iter = 142.
2023-05-30 17:39:51,894:INFO:

-------------Round number: 145 -------------


2023-05-30 17:39:54,561:INFO: Average Global Accurancy = 0.7884, Loss = 0.82.
2023-05-30 17:39:54,562:INFO: Best Global Accurancy = 0.7988, Loss = 0.78, Iter = 142.
2023-05-30 17:40:05,655:INFO:

-------------Round number: 146 -------------


2023-05-30 17:40:08,322:INFO: Average Global Accurancy = 0.7969, Loss = 0.79.
2023-05-30 17:40:08,322:INFO: Best Global Accurancy = 0.7988, Loss = 0.78, Iter = 142.
2023-05-30 17:40:19,401:INFO:

-------------Round number: 147 -------------


2023-05-30 17:40:22,065:INFO: Average Global Accurancy = 0.7809, Loss = 0.86.
2023-05-30 17:40:22,066:INFO: Best Global Accurancy = 0.7988, Loss = 0.78, Iter = 142.
2023-05-30 17:40:33,132:INFO:

-------------Round number: 148 -------------


2023-05-30 17:40:35,790:INFO: Average Global Accurancy = 0.7904, Loss = 0.82.
2023-05-30 17:40:35,791:INFO: Best Global Accurancy = 0.7988, Loss = 0.78, Iter = 142.
2023-05-30 17:40:46,836:INFO:

-------------Round number: 149 -------------


2023-05-30 17:40:49,492:INFO: Average Global Accurancy = 0.7944, Loss = 0.82.
2023-05-30 17:40:49,492:INFO: Best Global Accurancy = 0.7988, Loss = 0.78, Iter = 142.
```

4. Insights on studying Federated Learning

▼ The general description of the workflow in federated learning is as follow:

1. Initialization
    - Initialize a central server that coordinates the federated learning process.
    - Define the model architecture that will be used for training.
    - Establish a connection with participating edge devices, aka users or clients, that hold the local data, which is invisiable by the server machine.
2. Data Distribution
    - This step is completed by the code TA written. But the general idea is to distribute the model's initial parameters to all users.
    - And ensure that the devices have the necessary software and infrastructure to train the model locally.
3. Local Model Training
    - Each participating device independently trains the model using its local data.
    - Devices perform multiple epochs of training using their local optimization algorithms, which are Adam in this assignment.
    - During training, devices compute weights and gradients based on their local data and update their local model parameters
4. Model Aggregation
    - The devices then send their trained parameters, which is weight in this case, to the central server.
    - The central server then aggregates the received model parameters from the edge devices. (This is the function we had to completed)
    - Aggregation methods can vary, in this case we are simply aggregating the parameters by weights of the user samples.
5. Model Update and Iteration
    - The central server receives the aggregated model weights and updates the server.
    - The updated server is then redistributed to the participating edge devices for the next round of training.

6. Iterative Process
    - The process of Local Model Training till Model Update and Iteration are repeated for multiple rounds. (In some cases, we can set the conergence criteria, triggering to terminate the iteration process)
    - This iterative process allows the model to benefit from the collective knowledge of all participating edge devices while preserving data privacy.
7. Final Model
    - Once the federated learning process is completed, the final model is obtained by the weighted average of the models from all devices. (This assginment specifically, it can be other averaging techniques)
    - Finally, the central model can be deployed for inference or further evaluation.

▼ Final Comments

I really learned a lot from this assignment and class materials. By applying federated learning, we can address the growing concerns surrounding data privacy and confidentiality in today's digital age. This assignment has provided valuable insights into training models on decentralized data without compromising individual privacy. Through the experiments conducted on distribution and convergence speed, we have witnessed the potential of federated learning in preserving data privacy while achieving efficient and effective model training.

In addition to its privacy-preserving nature, federated learning has the advantage of leveraging the collective power of diverse devices. By allowing each device to train the model on its local data, we harness the increased diversity and create larger effective training sets. This diversity, coupled with the reduced biases achieved through aggregation, contributes to the potential for faster convergence and better model performance.

The combination of local model insights from multiple devices further enhances the learning process. Each device brings its unique perspective, capturing domain-specific knowledge and local patterns. By aggregating these insights, federated learning enables us to harness the collective intelligence and build models that are more robust, accurate, and representative of the overall data distribution.

Overall, this assignment has shed light on the practical applications and advantages of federated learning. By embracing this decentralized approach, we can strike a balance between data privacy and model performance, leading us into a future where advanced machine learning models can be trained while preserving individual privacy.