Data Science HW 4 – Horizontal Federated Learning

系級：理學院學士班 23 級

學號：108020017

姓名：黃珮綺

---

1. Code

■ Server-side

`def aggregate_parameters(self)`

This function aims to aggregate the user models and update the global model. We use a weight of the number of available training samples to users to summarize the parameters of the local models. The sum of the parameters will directly replace the original global model and serve as the new one in the next round.

```
56      def add_parameters(self, user, ratio):
57          for server_param, user_param in zip(self.model.parameters(), user.model.parameters()):
58              server_param.data = server_param.data + user_param.data.clone() * ratio
59
60      def aggregate_parameters(self):
61          assert (self.selected_users is not None and len(self.selected_users) > 0)
62
63          # Initialize parameters with zeros.
64          for param in self.model.parameters():
65              param.data = torch.zeros_like(param.data)
66
67          # Calculate the number of training samples which are currently used.
68          total_train = 0
69          for user in self.selected_users:
70              total_train += user.train_samples
71
72          # Calculate the weighted sum of local parameters.
73          for user in self.selected_users:
74              self.add_parameters(user, user.train_samples/total_train)
```

`def select_users(self, round, num_users)`

We will select `num_users` of users from the set of all available users to do the training, which is done in this function. It randomly chooses `num_users` of users out of the user set and returns the subset. Also, it will take the minimum of the total number of users and `num_users` to ensure that the number of selected users is always smaller than or equal to the total number of users.

```
102     def select_users(self, round, num_users):
103         # If num_users equals the number of total users, simply returns all users
104         if(num_users == len(self.users)):
105             print("All users are selected")
106             return self.users
107
108         # Otherwise, randomly choose {num_users} users from all users.
109         # Taking min{num_users, len(self.users)} to ensure that num_users is not larger
110         # than the total number of users.
111         num_users = min(num_users, len(self.users))
112         print("Select %d users" % (num_users))
113         return np.random.choice(self.users, num_users, replace=False)
```

■ Client-side

`def set_parameters(self, model, beta=1)`

This function is to update the parameters of the local model. The new parameters will sum up the global model parameters and the old local parameters with a ratio.

```
68      def set_parameters(self, model, beta=1):
69          for old_param, new_param in zip(self.model.parameters(), model.parameters()):
70              # Moving average model with beta
71              if beta == 1:
72                  old_param.data = new_param.data.clone()
73              else:
74                  old_param.data = beta * new_param.data.clone() + (1 - beta) * old_param.data.clone()
```

■ Plot accuracies and losses

In order to identify the convergence of the model, I add a function `plot()` in `serverbase.py`. The loss and accuracy of the server model will be stored in a python dictionary `self.metrics`, where we can access by keys `'glob_loss'` and `'glob_acc'`. This function will first make sure that the keys are contained in `self.metrics`, and then plot the accuracies and losses. The plot will be stored under the same directory as the result of the model.

```
233     def plot(self):
234         import matplotlib.pyplot as plt
235 
236         # Check if 'glob_loss' is contained in self.metrics
237         if 'glob_loss' not in self.metrics.keys():
238             print("Unable to plot loss")
239         else:
240             plt.figure('Loss')
241             plt.plot(self.metrics['glob_loss'], label='Loss per iter')
242             plt.xlabel('Epoch')
243             plt.ylabel('Loss')
244             plt.ylim([0, 3])
245             plt.legend(loc='lower right')
246 
247             # Save plot
248             fig_path = os.path.join(self.save_path, self.algorithm, "models", self.dataset)
249             if not os.path.exists(fig_path):
250                 os.makedirs(fig_path)
251             plt.savefig(os.path.join(fig_path, "loss.png"))
252 
253         # Check if 'glob_acc' is contained in self.metrics
254         if 'glob_acc' not in self.metrics.keys():
255             print("Unable to plot accuracy")
256         else:
257             plt.figure('Accuracy')
258             plt.plot(self.metrics['glob_acc'], label='Accuracy per iter')
259             plt.xlabel('Epoch')
260             plt.ylabel('Accuracy')
261             plt.legend(loc='lower right')
262 
263             # Save plot
264             fig_path = os.path.join(self.save_path, self.algorithm, "models", self.dataset)
265             if not os.path.exists(fig_path):
266                 os.makedirs(fig_path)
267             plt.savefig(os.path.join(fig_path, "accuracy.png"))
```

In `main.py`, a new argument `'--plot'` is added. If plots are requested, the server will call `plot()` to draw the plots of accuracies and losses per iteration, respectively, right after the server finish training and testing.

```
21  def run_job(args, i, logging):
22      torch.manual_seed(i)
23      print("\n\n          [ Start training iteration {} ]           \n\n".format(i))
24      # Generate model
25      server = create_server_n_user(args, i, logging)
26      if args.train:
27          server.train(args)
28          server.test()
29
30          # Plot the accuracies and losses per iteration.
31          if args.plot:
32              server.plot()
```

■ Run code

This assignment uses Google Colab supported with GPUs to train the model. The script and the result are provided in `ds_hw4.ipynb`.

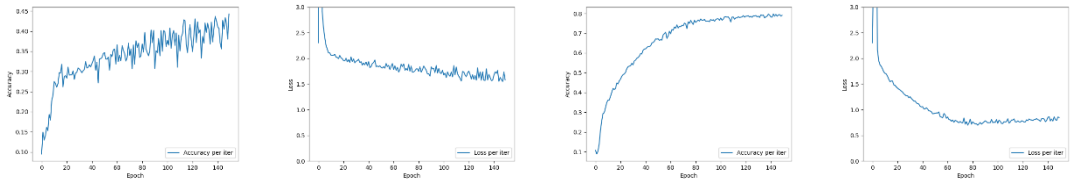
2. Performance Overview

2.1 Data distribution

In `generate_niid_dirichlet.py`, it repeats the raw data `alpha` times and uses a Dirichlet distribution to generate training data. In other words, a smaller alpha indicates lesser repetition and higher data heterogeneity.

Using the same model, `alpha`=0.1 converges early after 10 iterations and only reaches 0.4426 accuracies, whereas `alpha`=50 can jump out of local optimums and converges after 60 iterations with 0.7971 accuracies. This is possible because the training data of `alpha`=0.1 may be limited and the labels may be unbalanced.

Here, we discuss the data distribution provided in `ds_hw4.ipynb`. In CIFAR10, there are typically 10 classes in total. However, for the distribution with `alpha`=0.1, user [1] got 7517 training samples from only 5 identical classes. The training labels are also unbalanced among classes. For instance, Class 8 yields 3485 samples, while Class 5 yields only 4 samples. On the other hand, the distribution with `alpha`=50 has relatively balanced data and contains samples from all classes. As a result, it can keep finding global optimums and yield better model accuracy.

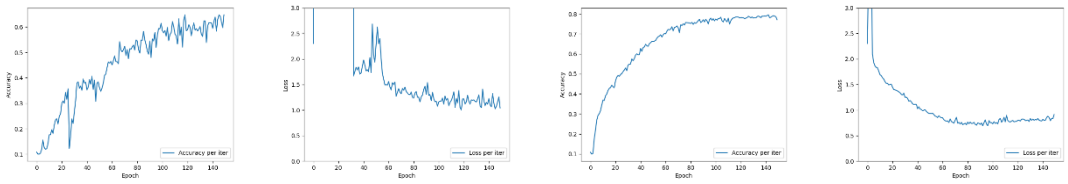


(a) Accuracy, alpha=0.1 (b) Loss, alpha=0.1 (c) Accuracy, alpha=50 (d) Loss, alpha=50

Figure1. Accuracy plots and loss plots with `alpha`=0.1 and `alpha`=50.0.

2.2 Number of users in a round

The final accuracy of the model with `num_users`=2 (=0.6482) is lower than that with `num_users`=10 (=0.7956). The possible reason is that `num_users` is too small. Since the global model will update its parameter by summing local parameters after each iteration, the local models can decide whether the global model is good or not. Particularly, lesser local model training in parallel can have a larger impact on the global model for each local one. If there is a local model that is badly trained, the global model will be easily spoiled. This results in significant oscillation in accuracy and loss. In contrast, the accuracy and loss curves are smooth when `num_user`=10.



(a) Accuracy, user=2 (b) Loss, user=2 (c) Accuracy, user=10 (d) Loss, user=10

Figure2. Accuracy plots and loss plots with `num_users`=2 and `num_user`=10.

3. Result

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


Select 10 users
Average Global Accurancy = 0.7899, Loss = 0.79.
Best Global Accurancy = 0.7915, Loss = 0.84, Iter = 137.
Finished training.
```

4. Discussion

In this homework, we implemented horizontal federated learning. We download the global parameters from the server and use the weighted sum of local parameters to update global parameters after training. Also, we enable concurrent training by assigning numbers of users to train locally in parallel. The experiments show that federated learning can be made practical with the following benefits: 1) `FedAvg` can train high-quality models using relatively few rounds of communication between server and users, and 2) selecting a fraction of clients to be trained concurrently can enhance computational efficiency and accuracy. Last but not least, 3) data privacy could be ensured by only sharing parameters during each round of communication. Moreover, we discuss the data heterogeneity. The federated learning paradigm enables efficient knowledge distillation without requiring any external data. This approach can benefit federated learning with better model performance on limited and unbalanced data.