1. 解釋程式碼

- select_user()：

```
def select_users(self, round, num_users):
    ## TODO
    '''
    Randomly select {num_users} users from all users
    Args:
        round: current round
        num_users: number of users to select
    Return:
        List of selected clients objects

    Hints:
        1. Default 10 users to select, you can modify the args {--n
        2. Note that {num_users} can not be larger than total users
    '''

    if(num_users == len(self.users)):
        return self.users

    num_users = min(num_users, len(self.users))

    np.random.seed(round)

    return np.random.choice(self.users, num_users, replace=False)
```

若輸入的 num_users 大於 self.users，就將全部的 self.users 回傳。反之，就隨機挑選 num_users 個 users 回傳。

- aggregate_parameters()：

```
def aggregate_parameters(self):
    ## TODO
    '''
    Weighted sum all the selected users' model parameters by number of samples

    Args: None
    Return: None

    Hints:
        1. Use self.selected_users, user.train_samples.
        2. Replace the global model (self.model) with the aggregated model.
    '''
    for param in self.model.parameters():
        param.data = torch.zeros_like(param.data)
    total = 0
    for user in self.selected_users:
        total += user.train_samples
    for user in self.selected_users:
        model = self.model.parameters()
        for server_param, user_param in zip(self.model.parameters(), user.get_parameters()):
            server_param.data = server_param.data + user_param.data.clone() * (user.train_samples / total)
```

首先將原本 server 的 model parameters 清 0，方便之後直接拿 users 的 parameters 取代。再來就將每個 users 的 parameters 乘上「users 所使用的 train_samples 在全部 samples 當中所佔的比例」，並加到 server 的 model parameters 當中，完成 aggregate。

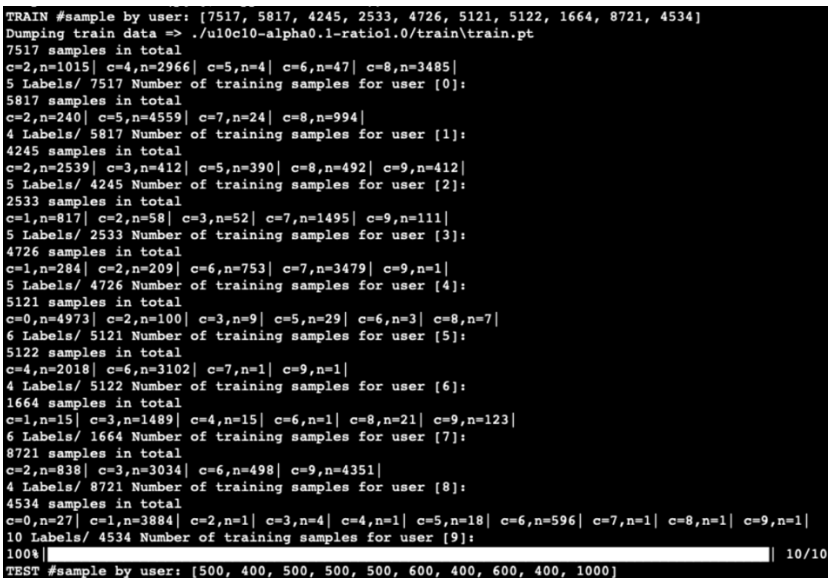
- set_parameters():

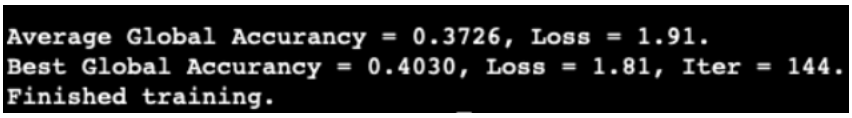
```
def set_parameters(self, model, beta=1):
    ## TODO
    '''
    Replace the user's local model with the global model
    Args:
        model: the global model parameters
        beta: moving average model,
            i.e., user's model parameters = beta * global model parameters + (1 - beta) * user's mc
    Return:
        None

    Hint:
        1. You can use self.model (the user's model), model (global model parameters).
    '''

    for user_param, global_param in zip(self.model.parameters(), model.parameters()):
        user_param.data = beta * global_param.data.clone() + (1 - beta)  * user_param.data.clone()
```

根據 beta，將 global parameters 與 local parameters 做結合。結合的方式為：beta *

global model parameters + (1 - beta) * local model parameters。

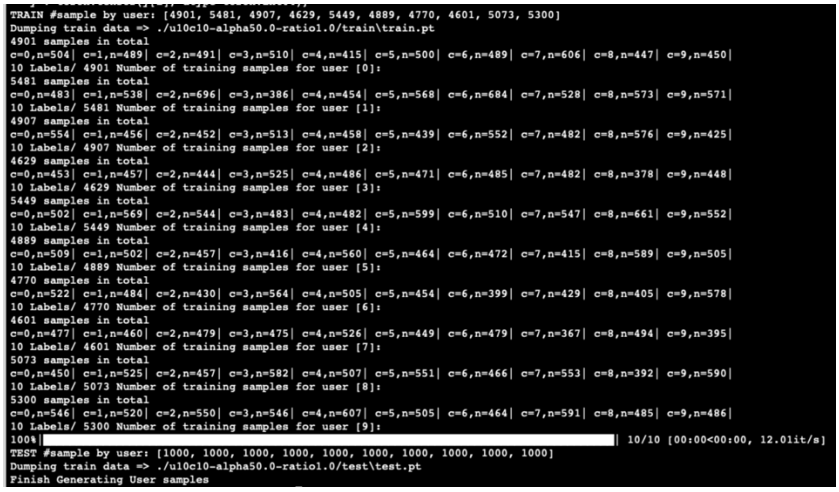
2. 探討問題
   I. data distribution
      a. alpha=0.1

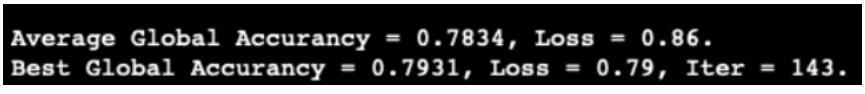




當 alpha 為 0.1 時，資料在每個 user 上面分佈的非常不平均，所以的 users 都只有特定 label 的資料，導致每個 local model 都無法訓練的非常全面，無法平衡地學習到全部的 features，因此也導致最後的 global accuracy 只有 0.3 左右。

      b. alpha=50





當 alpha 為 50 時，資料在每個 user 上面分佈的相對平均，所以的 users 都擁有

全部 label 的資料，因此每個 local model 可以平衡地學習到全部的 features，最後的 global accuracy 也達到 0.78 左右。

II. Number of users in a round

a. num_users=2

```
Average Global Accurancy = 0.6084, Loss = 1.19.
Best Global Accurancy = 0.6680, Loss = 0.96, Iter = 145.
Finished training.
```

b. num_users=10

```
Average Global Accurancy = 0.7869, Loss = 0.82.
Best Global Accurancy = 0.7955, Loss = 0.80, Iter = 146.
Finished training.
```

資料在每個 user 上面分佈的相對平均的情況下，可以看到每一輪更新 10 個 users 對 global accuracy 有著更想顯著的幫助。這是因為每個 users 都可能擁有獨特的數據，因此綜合每個 users 的參數更新都可以讓 global model 在整體上學習的更好。至於收斂的速度，我的實測結果為這兩種訓練方式都在 120~130 個 epochs 收斂。

3. 最終 acc 的輸出

```
Average Global Accurancy = 0.7869, Loss = 0.79.
Best Global Accurancy = 0.7988, Loss = 0.77, Iter = 144.
Finished training.
```

4. 此次作業中學到的重點

在 trace code 的過程當中，使我對 federated learning 有更深刻的了解。

a. 我了解到 server 如何選擇要更新的 user model，也學習到 server 與 users 如何根據收到的參數，更新自身的 model。

b. 並了解每個 users 所擁有的資料分布，對於最後的 global accuracy 會有很大的影響。server 需要適應不同的數據分佈，並綜合考慮各個使用者的貢獻，以獲得更好的全域準確度。

c. 每一輪更新的 users 數量，對 global accuracy 也會有影響。如果每輪僅選擇少數 users 與模型更新，可能會造成部分 users 的資料未被充分利用，導致模型在表現不佳。