Report

- Code Explanation

```python
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 {--num_users} to change this hyper-parameter
        2. Note that {num_users} can not be larger than total users (i.e., num_users <= len(self.user))
    '''
    assert num_users <= len(self.users)

    selected_indices = np.random.choice(len(self.users), num_users, replace=False)
    self.selected_users = [self.users[i] for i in selected_indices]
    return self.selected_users
```

首先修改的是 select_users，因為 num_users 需要小於等於 len(self.user)，所以我在一開頭增加了 assert 避免錯誤，隨後使用 np.random.choise 隨機取出 user 的 index，再透過 selected_indices 取出被選到的 user。

```python
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.
    '''
    now_params = list(self.model.parameters())
    total_samples = 0
    new_params = [torch.zeros_like(param) for param in now_params]

    for user in self.selected_users:
        user_params = list(user.model.parameters())
        total_samples += user.train_samples
        for user_param, new_param in zip(user_params, new_params):
            new_param.data += user_param.data * user.train_samples

    for now_param, new_param in zip(now_params, new_params):
        now_param.data.copy_(new_param.data / total_samples)
```

隨後修改的是 aggregate_parameters，因為需要計算個別 user 所擁有的 data 的比例，所以在迴圈過程中順便累加 total_samples，在最後將 new_param 的數值再去除以 total_samples 就可以得到正確的比例。在此處需要注意的

是，需要使用.copy_才可以正確的將值修改進模型參數內。

```
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 model parameters
    Return:
        None

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

    with torch.no_grad(): # avoid calculate gradient      You, 44 seconds ago • Uncommitted changes
        for now_param, global_param in zip(now_params, global_params):
            now_param.data.mul_(1 - beta).add_(beta * global_param.data)
```

最後修改的就是 set_parameters，該 function 的思路和 aggregate_parameters 很像，而且還簡單上一些，基本上就是根據所給出的公式，取出對應的 parameters 與 beta 做相乘再相加，即可得到新的 parameters，此處同樣要注意是否有將值修改進模型參數內，所以我對 data 直接使用運算函式將值修改進去。

- Data Distribution
  - alpha 0.1:

```
TRAIN #sample by user: [7517, 5817, 4245, 2533, 4726, 5121, 5122, 1664, 8721, 4534]
```

從圖中可以看得出來，10 個 user 的 samples 分佈很不平均，最多的有 8721，而最少得只有 1664。這導致擁有比較少 samples 的 user model 會很難增進，並且因為 data 可能沒有分佈均勻，讓模型只能更新局部，也會很難對 global model 有所貢獻，那麼在 global model 的最終表現就會比較不符合預期。以下是最終的模型表現：

```
Average Global Accurancy = 0.4122, Loss = 1.60.
Best Global Accurancy = 0.4270, Loss = 1.56, Iter = 147.
Finished training.
```

  - alpha 50:

```
TRAIN #sample by user: [4901, 5481, 4907, 4629, 5449, 4889, 4770, 4601, 5073, 5300]
```

相比 0.1，alpha=50 的情況下，各個 user 所擁有的 data 就比較平均，這將讓每一個 user model 都可以較為相等的對 global 進行更新，而這種類似於 ensemble 的概念也可以增進模型的 robustness，使得最終表現會比較好。以下是最終的模型表現：

```
Average Global Accurancy = 0.7988, Loss = 0.78.
Best Global Accurancy = 0.7988, Loss = 0.78, Iter = 149.
Finished training.
```

- Number of users in a round

- user 10

```
Average Global Accurancy = 0.7901, Loss = 0.82.
Best Global Accurancy = 0.7931, Loss = 0.83, Iter = 146.
Excution time: 23m11s
Finished training.
```

當 user=10 的時候，相當於每次更新都會將全部 user 進行更新，這種更新的方式讓每個 user 在每次 global iter 結束後都能跟上 global model 的腳步，讓其在訓練的過程中呈現一個比較平穩的下降，並且最終收斂的效果也會比較好。

- user 2

```
Average Global Accurancy = 0.6172, Loss = 1.16.
Best Global Accurancy = 0.6848, Loss = 0.92, Iter = 139.
Excution time: 16m20s
Finished training.
```

每一次 global iter 只更新兩個 user 的時候，整體的執行速度上僅有 16m20s，比起 user=10 也就是全部更新的情況快上很多，但在訓練的過程中，因為每次只有更新兩個 user 並且再對 global 做更新，使得模型的學習過程比較震蕩，這是因為 global model 的更新比較容易受到某些 user model 的影響，在 los 的下降過程中會走向比較 local minimum 的方向，那麼在最後收斂的效果就沒有 user10 來得好。

- Conclusion

從上課的影片中，我對於 federated learning 有了基本的觀念，並且老師也有進階為我們介紹 paper 的實作方法，讓我在這個主題上也有著更深刻的領悟。而在這次作業中，我們實作的比較基礎的 horizontal 版本，透過每個 user 更新後的參數，global model 根據比例去更新自身的參數，再將更新後的參數回傳給各個 user，讓 user 可以進行 moving average 的更新。這樣的過程可以將每個 user model 所學到的融入到 global，並且透過 global 的回傳，各個 user 也可以學到其他 user 所學到的。