HW5 Report

109000205 蕭皓隆

1. 程式碼實作

serverbase.py

```python
def calculate_samples(self):
    count = 0
    for user in self.selected_users:
        count += user.train_samples  # count all the samples

    return count

def new_parameters(self):
    new_parameters = {}
    for name, param in self.model.state_dict().items():
        new_parameters[name] = torch.zeros_like(param, dtype=torch.float32) # initialize the size of the parameters same as the original one

    return new_parameters

def aggregate_parameters(self):
    total_samples = self.calculate_samples() # count for the number of total samples
    new_parameters = self.new_parameters() # initialize the new parameters

    for user in self.selected_users:
        user_parameters = user.model.state_dict() # load current parameters in the model
        cur_weight = user.train_samples / total_samples # update the weight of current user

        for p in new_parameters:
            new_parameters[p] += cur_weight * user_parameters[p] # calculate new parameters

    self.model.load_state_dict(new_parameters) # update parameters in the model

    def select_users(self, round, num_users):
        if num_users <= len(self.user): # check the value is valid or not
            if round % 5 == 0:  # randomly select users every five rounds
                return random.sample(self.users, num_users)
            else:
                # select the users based on their size of model parameters
```

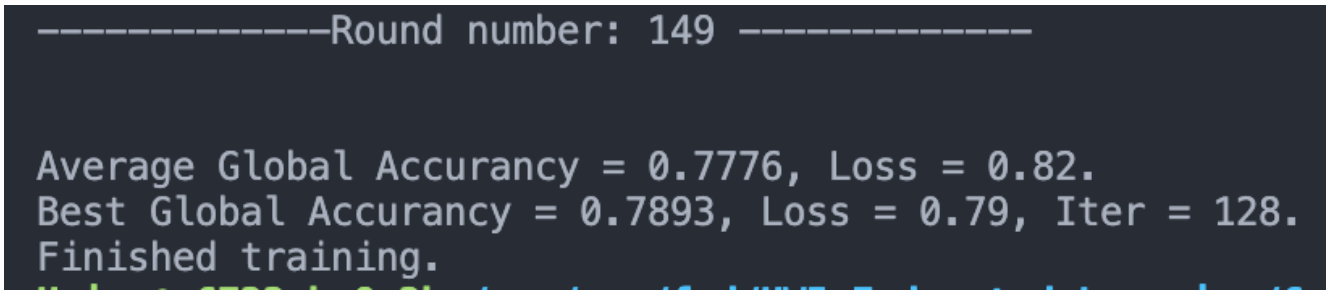
```
                sorted_users = sorted(self.users, key=lambda user:
len(user.model.state_dict()), reverse=True)
                return sorted_users[:num_users]
        else:
            # raise exception if the value of num_users is invalid
            raise Exception
```

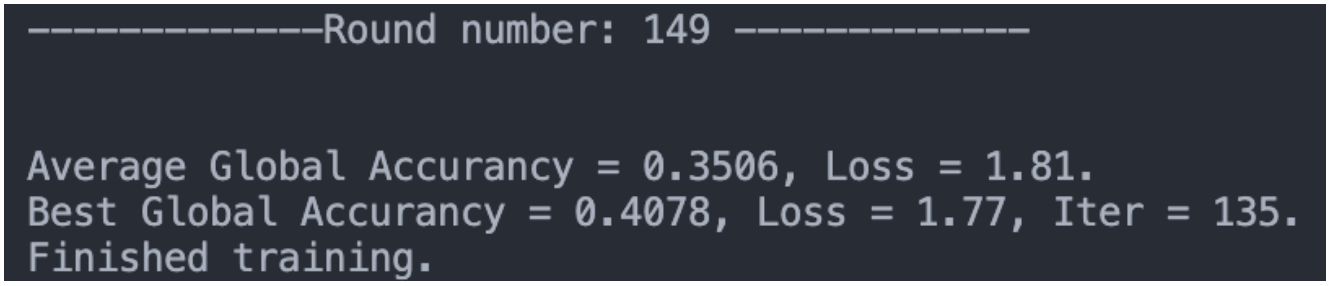
`userbase.py`

```
def set_parameters(self, model, beta=1):
  # iteratively update the user parameters by the definition
  for user_parameters, global_paramers in zip(self.model.parameters(),
model.parameters()):
      user_parameters.data = beta * global_paramers.data + (1-beta) *
user_parameters.data
```

2. 問題探討

Data distribution

- 在`generate_niid_dirichlet.py`中，$alpha$決定Dirichlet distribution中分佈的集中性，較小的$alpha$會讓users資料分布較集中、差異性較小；較大的$alpha$則會讓users資料分布較發散、差異性較大。由結果可知，$alpha = 50$的情況下可以有較好的global model accuracy。
- $alpha = 50$
  - 

- $alpha = 0.1$
  - 

Number of users in a round

- num_users決定了參與訓練的user數量。在$num_users = 2$的情況下，global model accuracy一直無上升的趨勢，且收斂速度較慢，雖然Loss有下降，但仍保持在較大的值；在$num_users = 10$的情況下，global model accuracy會逐漸上升，收斂速度比起來明顯較快。
- $num\_users = 2$
  - ```
    -------------Round number: 149 -------------


    Average Global Accurancy = 0.3001, Loss = 2.56.
    Best Global Accurancy = 0.4310, Loss = 1.59, Iter = 77.
    Finished training.
    ```
- $num\_users = 10$
  - ```
    -------------Round number: 149 -------------


    Average Global Accurancy = 0.7965, Loss = 0.78.
    Best Global Accurancy = 0.8006, Loss = 0.76, Iter = 145.
    Finished training.
    ```

3. Model Accuracy

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


  Average Global Accurancy = 0.7921, Loss = 0.82.
  Best Global Accurancy = 0.8016, Loss = 0.78, Iter = 137.
  Finished training.
  ```

4. 學到的重點

- 在這次作業中，嘗試訓練到以往只學習到觀念但較少實作的聯邦式學習。過程中，從user的選擇到global model參數更新的權重，都是可以著手優化並影響performance的重點，雖然皆是使用程式碼模擬，但大致上的流程與架構都讓我對聯邦式學習有更進一步的認知。
  ```