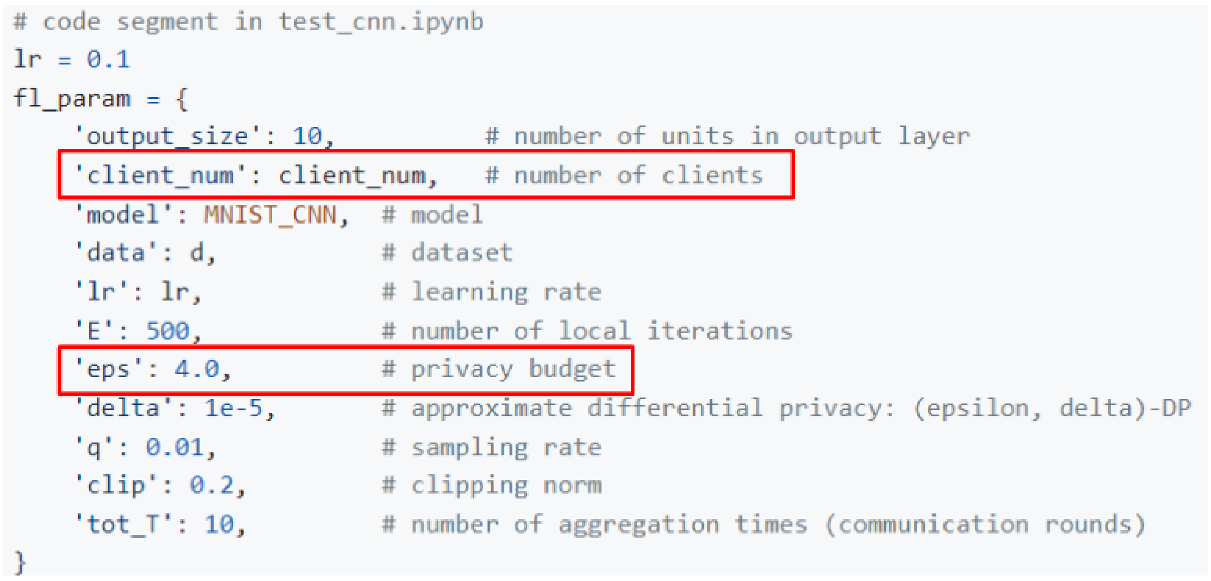
Distributed Differential Privacy Applied in Federated Learning

最後一組

M11215032 葉品和 M11215052 陳奕帆 M11215066 鄭宜珊

Dataset : Mnist

1.How do you perturb the data?



```
# code segment in test_cnn.ipynb
lr = 0.1
fl_param = {
    'output_size': 10,          # number of units in output layer
    'client_num': client_num,   # number of clients
    'model': MNIST_CNN,  # model
    'data': d,           # dataset
    'lr': lr,            # learning rate
    'E': 500,            # number of local iterations
    'eps': 4.0,          # privacy budget
    'delta': 1e-5,       # approximate differential privacy: (epsilon, delta)-DP
    'q': 0.01,           # sampling rate
    'clip': 0.2,         # clipping norm
    'tot_T': 10,         # number of aggregation times (communication rounds)
}
```

(1) ‘client_num’ sets the number of clients: n.

(2) ‘eps’ sets the privacy level ε.

Divided into FLClient and FLServer, each client will get the same model from the Server and unify the initialization parameters of the model. Repeat the following steps to train the model：

```
class FLClient(nn.Module):
    """ Client of Federated Learning framework.
        1. Receive global model from server
        2. Perform local training (compute gradients)
        3. Return local model (gradients) to server
    """
```

```
class FLServer(nn.Module):
    """ Server of Federated Learning
        1. Receive model (or gradients) from clients
        2. Aggregate local models (or gradients)
        3. Compute global model, broadcast global model to clients
```

a. Each Client uses its own data to train the model, calculates its own gradient, and then uploads it to the Server.

```
# update local model
optimizer.step()
```

b. The Server integrates the gradients of each Client and updates the model.

```
def aggregated(self, idxs_users):
    """FedAvg"""
    model_par = [self.clients[idx].model.state_dict() for idx in idxs_users]
    new_par = copy.deepcopy(model_par[0])
```

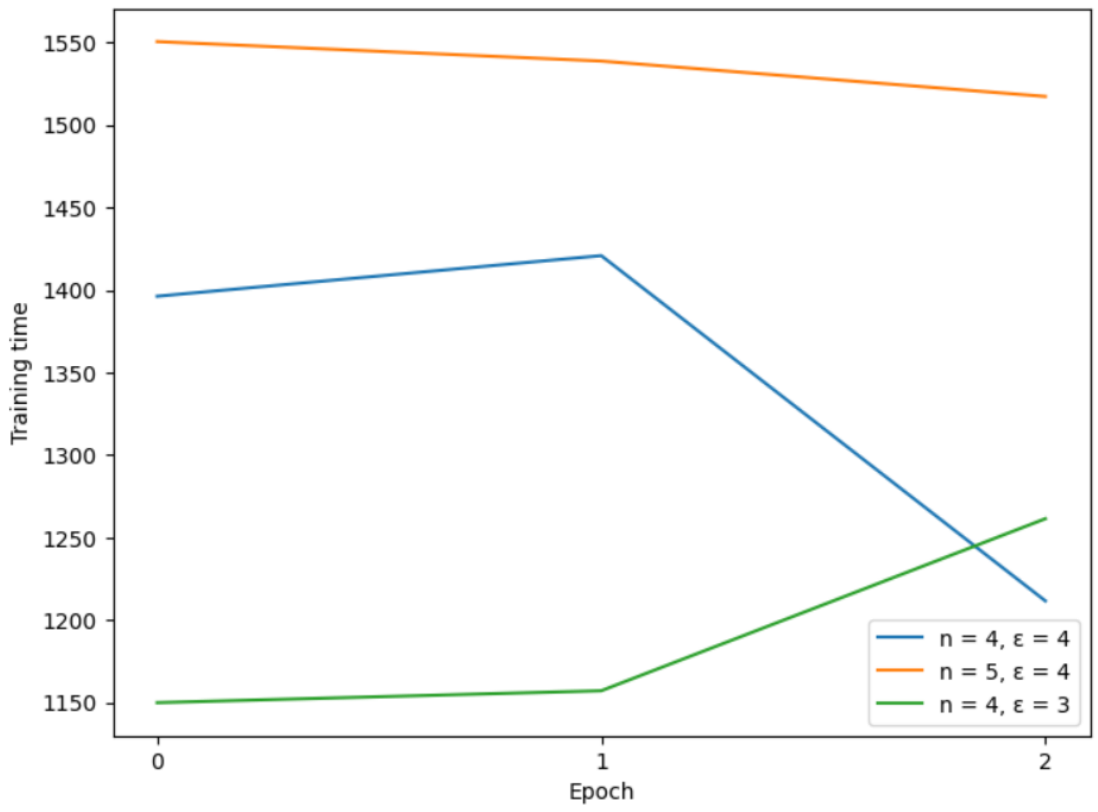
c. Server returns the updated gradient of the model to each Client.

```
def broadcast(self, new_par):
    """Send aggregated model to all clients"""
    for client in self.clients:
        client.recv(new_par.copy())
```

d. Clients update their respective models.

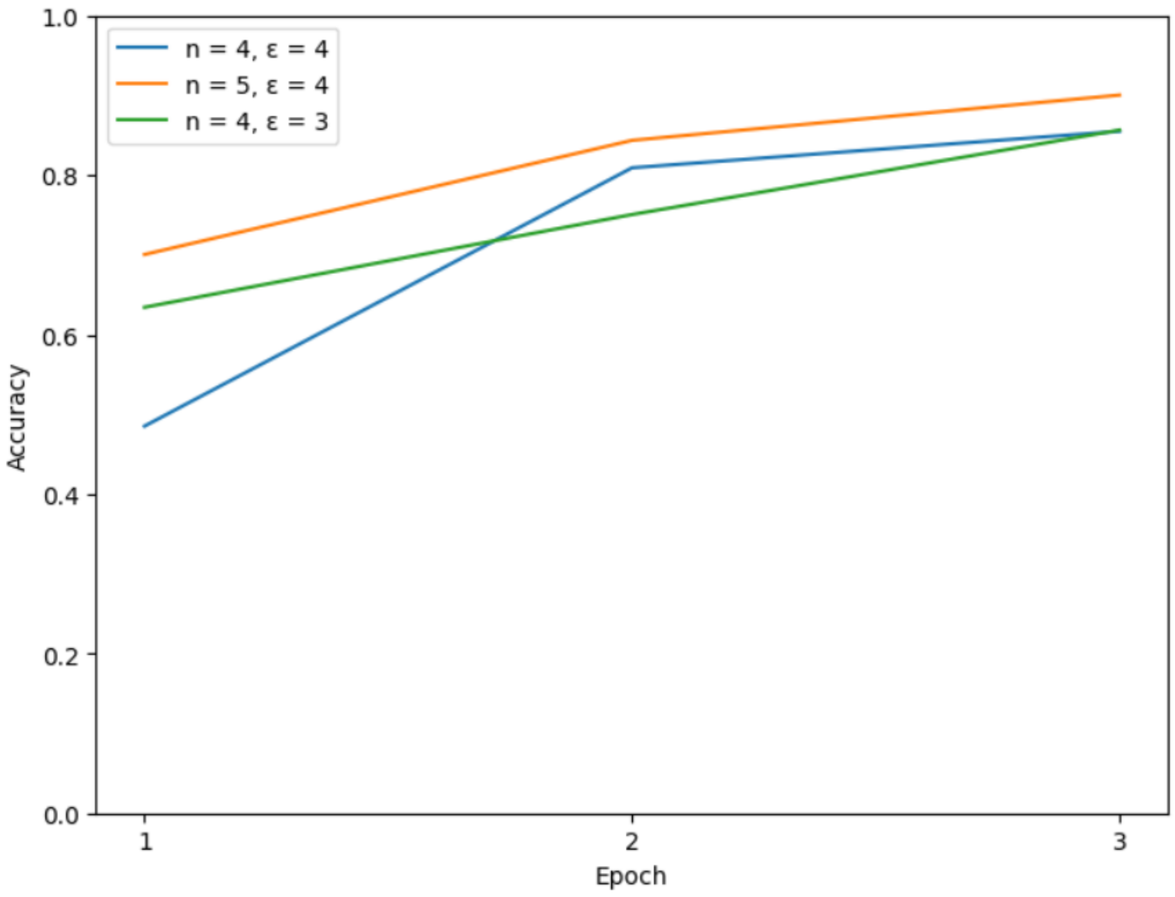
```
def update(self):
    """local model update"""
    self.model.train()
    criterion = nn.CrossEntropyLoss(reduction='none')
    optimizer = torch.optim.SGD(self.model.parameters(), lr=self.lr, momentum=0.9)
    # optimizer = torch.optim.Adam(self.model.parameters())
```

Since the training time of each epoch is very long, we do not train too many epochs. The following figure shows the training time of each epoch.



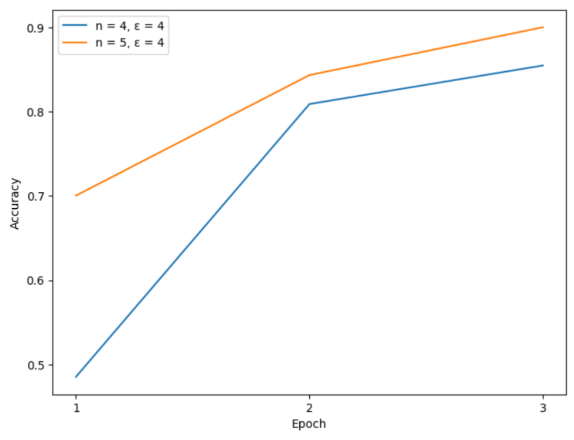
2.Effectiveness measure: Accuracy

- learning rate = 0.15
- epoch = 3
- delta = 1e-5
- sampling rate = 0.01

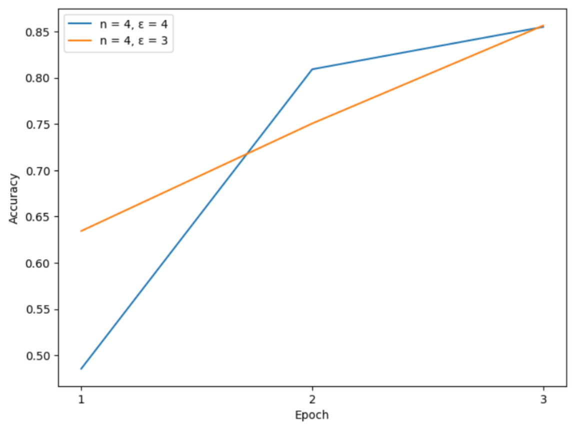


3.Privacy level: the value of ε & Number of clients: n

- learning rate = 0.15
- epoch = 3
- delta = 1e-5
- sampling rate = 0.01



ε = 4



n = 4