Environment： pytorch, torchvision,numpy,pandas,

Cifar10：

fail_train_IID_cifar10.py 和 fail_train_nonIID_cifar10.py。

Mnist：

fail_train_IID_mnist.py 和 fail_train_nonIID_mnist.py。

take mnist/IID as an example，

IID， execute python fail_train_IID_mnist.py

Result:

```
model model_1 upload weights
[g_epoch 22]: model(model_2) local training
        [model model_2][local epoch 0]:train loss = 0.004349, acc = 0.998496, val loss = 0.233893 val acc = 0.968750
        [model model_2][local epoch 1]:train loss = 0.003942, acc = 0.998692, val loss = 0.235151 val acc = 0.968963
        [model model_2][local epoch 2]:train loss = 0.003318, acc = 0.999019, val loss = 0.236408 val acc = 0.968657
        [model model_2][local epoch 3]:train loss = 0.002837, acc = 0.999232, val loss = 0.236558 val acc = 0.968628
        [model model_2][local epoch 4]:train loss = 0.002458, acc = 0.999385, val loss = 0.236293 val acc = 0.968625
[g_epoch 22]: model(model_2) local training finish, loss = 0.236293
model model_2 upload weights
[g_epoch 22]:model server update global model weights
[g_epoch 23]: local model model_0 download global model weights
[g_epoch 23]: cal adaptive data sampling lambda
[g_epoch 23]: local model model_0 upload local lambda
[g_epoch 23]: local model model_1 download global model weights
[g_epoch 23]: cal adaptive data sampling lambda
[g_epoch 23]: local model model_1 upload local lambda
[g_epoch 23]: local model model_2 download global model weights
[g_epoch 23]: cal adaptive data sampling lambda
[g_epoch 23]: local model model_2 upload local lambda
[g_epoch 23]: model(model_0) local training
        [model model_0][local epoch 0]:train loss = 0.004080, acc = 0.998661, val loss = 0.288079 val acc = 0.968750
        [model model_0][local epoch 1]:train loss = 0.002840, acc = 0.999267, val loss = 0.283071 val acc = 0.969116
        [model model_0][local epoch 2]:train loss = 0.002634, acc = 0.999384, val loss = 0.289104 val acc = 0.968831
        [model model_0][local epoch 3]:train loss = 0.002620, acc = 0.999378, val loss = 0.289435 val acc = 0.968700
        [model model_0][local epoch 4]:train loss = 0.002340, acc = 0.999464, val loss = 0.287697 val acc = 0.968905
[g_epoch 23]: model(model_0) local training finish, loss = 0.287697
model model_0 upload weights
[g_epoch 23]: model(model_1) local training
        [model model_1][local epoch 0]:train loss = 0.002831, acc = 0.999222, val loss = 0.285133 val acc = 0.967742
```

At round 23, the accuracy of each model reached to 97%。

Parameters

number of models:

n_models

```
91  models = []
92  n_models = 3
```

The number of local training rounds for each client independently before communication

```
120  ## pretraining each other
121  local_train_max_epoch = 2
122
```

Number of rounds of federal training:

```
132  ## FAIL Training
133  g_max_epoch = 50
```

In each round of federated training, the number of times each client trains itself locally

local_epochs

```
    for model_no,(model_name,model) in enumerate(models):
        traindata[model_name].setLambda(global_lambda,global_loss[model_name])
        #3 local training
        print("[g_epoch %d]: model(%s) local training"%(g_epoch,model_name))
        loss = train(model_name,model,traindata[model_name],testdata[model_name],g_epoch,local_epochs = 5,lr=1e-4)
        print("[g_epoch %d]: model(%s) local training finish, loss = %f"%(g_epoch,model_name,loss[-1]))
        model_server.upload_local_weight(model_name,model.state_dict(),loss)
        # print("[g_epoch %d]: model(%s) upload local weights"%(g_epoch,model_name))

```

Gpu,If you have a graphics card, turn it on to speed up local training data exponentially!

```
USE_CUDA = False

```