Federated Learning Report

SID 490157303 **SID 480399995**

1 Introduction

In this assignment, we implemented an instance of the *Federated Learning* algorithm which includes one server and five clients. We consider *MNIST* as out dataset, and use *Multinomial Logistic Regression* as the classifier. To simulate the distributed system, we run each clients and server in separate terminals, and the communication between server and clients is done by socket with UDP protocol.

2 Implementation Details

Our implementation is separated into three parts, which are *Server*, *Client* and *Classifier*. At the beginning, the server should be first started, and then each clients should be run. The whole federated learning system will run 100 iterations. In each iteration, the clients will test and train the received global model using its own testing and training data, and the server will listen and aggregate the local models from clients to build the new global model and then broadcast the new global model to each clients.

2.1 Classifier

The classifier we choose to use is *Multinomial Logistic Regression* which is a generalized logistic regression that is able to do multiclass classification. When the server is initialized, a multinomial logistic regression model will be created with random weights and biases. After the clients are started, the federated learning system will start the training process based on this initial model and the model would be updated after each training iteration.

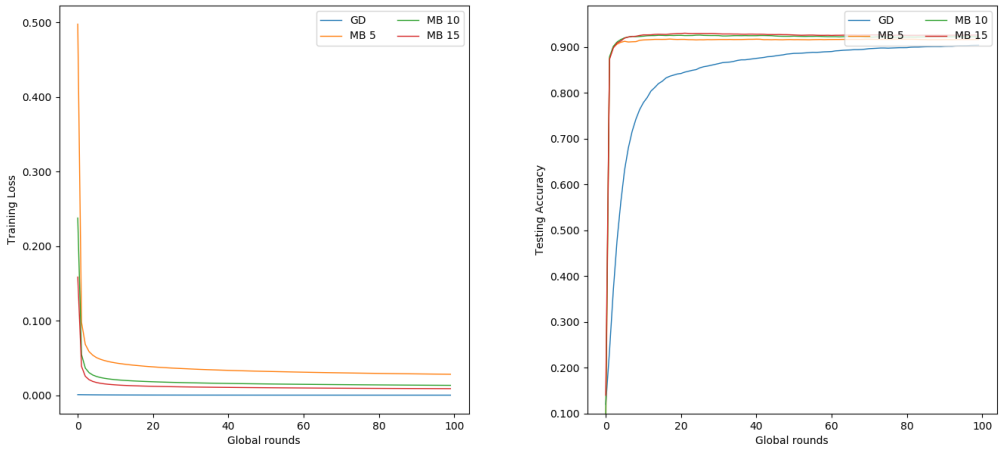
2.2 Server

The server should be the first program to run when starting the federated learning system. After the server starting to run, it will first try to detect the clients. It will keep waiting until it receives the first handshaking message from one of the clients. And then it will wait 30 seconds for the next handshaking message and so on until there is no message received in 30 seconds. The handshaking message is a string that contains the client-id and training data size of that client. For each handshaking message the server received, it will register the corresponding client. Then the server will generated a multinomial logistic regression model randomly and then broadcast it to all the registered clients.

After the initialization, the server will start to loop 100 iterations. In each iteration, it will first listen to the clients and store the local models sent by clients. Then server will use all or 2 of thoses models (based on the running commands) to aggregate the new global model. After the aggregation, the server will broadcast the new global model to all the clients no matter whether their local models were used in the aggregation.

During this process, there might be new clients send handshaking message. The server will register the new clients and send global model in next broadcasting.

In the end, the server will send a shut down message to all the clients and the federated learning system will be terminated.



(a) Training Loss (b) Testing accuracy

Figure 1: Training loss and testing accuracy of model that aggregating all local models

2.3 Client

The client should be start after the server. After a client start running, it will first loads its corresponding dataset and send handshaking message which includes its own identification and size of training data to server. Then the client will iteratively run the evaluating and training process until it receives the shut down message from server. Each iteration includes receiving global model, evaluating global on local dataset, training the global model, logging results and sending back the trained model to server.

2.4 Communication

The communication between server and client is done by socket with UDP protocol. Both server and client have two sockets, one is for sending the model and the other is for receiving the model.

There are 7,850 parameters (including weights and biases) in the multinomial logistic regression model, which is too large to send in one single message through UDP protocol. In the server end we break the model into multiple small packets and send each packet to clients separately. The packets will be aggregated in the client end once the client receives all the packets. The communication is very fast, accodring to our implementation, it takes only around 0.05 seconds for server to send model to all 5 clients.

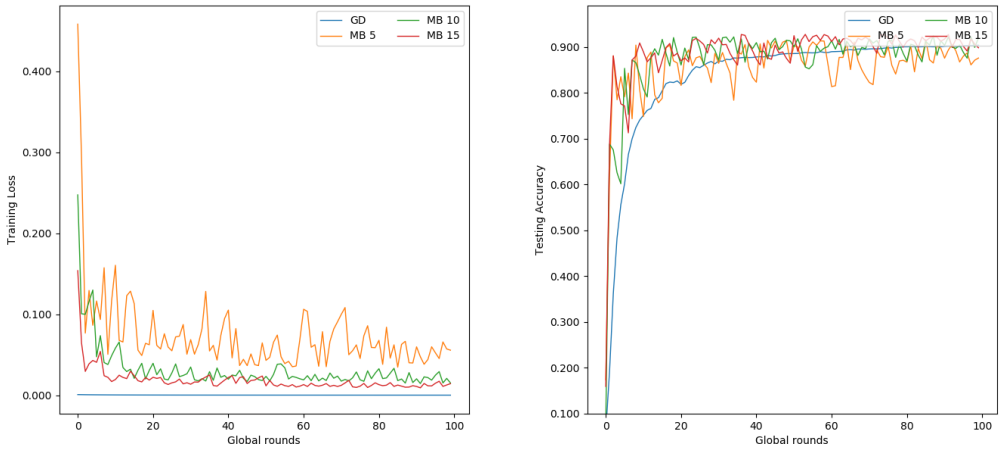
3 Performance Comparison

To evaluate the performance of the federated learning system, we carried out two groups of comparisons which are the comparison between using gradient descent and mini-batch (with batch size 5, 10, 15) gradient descent and the comparison between 2 subsampling clients and no subsampling clients.

3.1 GD vs Mini-Batch GD with different batch size

We run our federated learning system by using gradient descent and mini-batch gradient descent with batch size 5, 10, and 15. The results are shown in the Figure 1.

From the Figure 1 (a) we can see that the training loss of both gradient descent and mini-batch gradient descent are decreasing. The training loss of gradient descent looks like a horizontal line due to the scale problem of the graph, but its value is indeed decreasing (Please refer to the client log to check). While the training loss of each groups decreasing, their corresponding average testing



(a) Training Loss

(b) Testing Accuracy

Figure 2: Training loss and testing accuracy of model that aggregating 2 subsampling local models

accuracies are increasing as shown in Figure 1 (b). We can see that in the first 5 iterations, the training loss decreased sharply and the testing accuracy increased sharply. This means the system is learning rapidly in this peroid of time. After 10 iterations, the changes in both slowed down.

Figure 1 (b) clearly shows that the testing accuracy of mini-batch stochastic gradient descent with batch size 15 is the highest in after training for 100 iterations. We can see that for mini-batch gradient descent, while the batch size increasing, the corresponding accuracy is increasing as well. However, since 5, 10 and 15 are relatively small batch size compared with the number of samples in the whole dataset, we cannot conclude that the testing accuracy will always increase as the batch size increase. But we can say that, when batch size is relatively small, larger batch size will give us better testing accuracy.

From the figure we can also see that after 100 iterations, the gradient descent have a little bit lower testing accuracy than the mini-batch gradient groups. This means that mini-batch gradient descent learned faster than the gradient descent.

When it somes to the running time, we realized that the mini-batch gradient descent runs a lot faster than the gradient descent. So if we would like to training a model rapidly we can choose the mini-batch gradient descent.

3.2 Subsampling vs No subsampling

According to Figure 2, the training loss of both gradient descent and mini-batch gradient descent are roughly on a downward trend while their corresponding testing accuracy are roughly on a upward trend. Compared with the group that aggregate the local models of all clients (as shown in Figure 1), there are a lot fluctuate in Figure 2. This is because that we only randomly pick two out of five clients to aggregate their local model, which means the parameter changes in other clients will be ignored when building the global model.

However, if we only ask the chosen clients to send their local model to the server, subsampling clients can contribute on saving communication cost. Since the multinomial logistic regression model is not a very huge model, the subsampling clients strategy does not have significant positive influence on the communication cost and even has negatvie influence on model stability. But when people trying to train a model with large scale parameters by federated learning, subsampling clients would save tons of time.