November 2021 Review

Emmanouil Petrakos

Server - Client System

The Federated Learning training is set up as cross-device. The server holds minimum information about the clients, just enough to maintain connection and communicate. As a result, the server can not address clients directly and no confirmations are conveyed on any direction.

Each global epoch (GE), the server announces the global model to all connected clients. Clients updates the model locally using a part of their data and send back to the server the new trainable variables or their deltas. Finally, the server aggregates the local models and computes the new global model through Federated Averaging. This operation continues until the ending requirements are met, e.g. a desired accuracy was reached or a certain number of epochs passed.

The server, in addition to orchestrating the training and and calculating the global model, can import a pretrained model and save the final one. Also, in the first GE the clients can ignore the global model if it is not pretrained. Instead they can use their own pretrained or initialized model.

Server Overview

Before FL starts, the server has to complete three tasks. Load a pretrained model if it exists, set up a listening socket to receive incoming connections and create a structure to orchestrate the FL training. This structure follows the event driver TCP server paradigm. A single process, handles every event (accept a connection, data available to read, announce new global model) on a callback. [1]

An event can be raised either by the listening socket or by a connected one. In the first case, the connection is accepted and a new socket is created and inserted in the event structure. Also, the server responds to the new client with the current global model in order to let him start working immediately. In the second case, the corresponding client either disconnected or data was received. If the former is true, the socket is removed from the event structure and destroyed. If the latter is true, the received data is temporary stored to a buffer designated for that connection. This need to happen because a message could be greater than the operating system's socket windows and multiple read events would be raised. When a message is fully received, if valid, its model is aggregated to a cumulative model of the local ones.

After any event, the requirements for a new epoch are checked. Their parameters are how many local models are successfully received, how many clients are connected and how many clients are still working. Provided they are satisfied, a new global model is created and broadcasted to all clients. The new model is calculated by dividing the cumulative model by the number of the local models that were used to create it.[2] Furthermore, the ending requirements are checked. If they are fulfilled, the final model is send to the clients in order to test its accuracy.

Client Overview

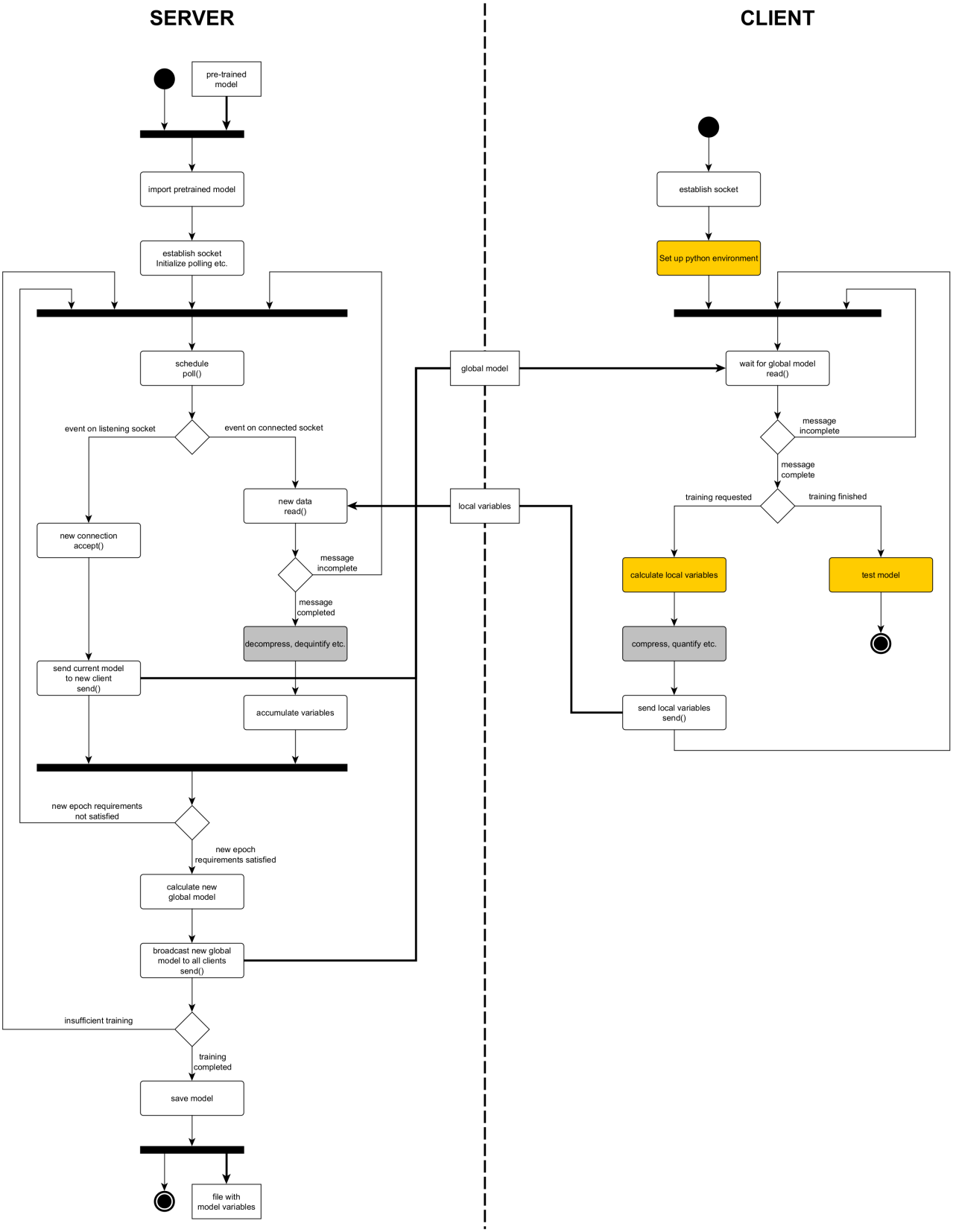
At start up, the client needs to complete two tasks. First, set up the training environment. As training is achieved with Tensorflow in Python and the rest of the codebase is developed in C++, the Python interpreter must be embedded before the model is compiled. The second task is to create a socket and connect to the FL server.

After the necessary initializations are completed, the client waits to receive a message by the server. When this happen, the model variables are passed to the Python environment and inserted in the local model. After a predefined number of local training, the new local model variables are exported back and used to create a reply to the server. As the model variables are transmitted in C-like arrays, they can not be directly used in the Python environment. To circumvent that, NumPy array metadata are created around them.

[1] Till now, the server's computations does not seem to add any significant delay. In case that stops being truth, the structure can be modified to assign a thread per client or use a pool of worker threads.

[2] In case deltas were used, the cumulative model is added to the previous global model.

Server - Client Activity Diagram



Error states are not showed. Yellow states are modelled with TensorFlow and grey are unimplemented.

Communication Scheme

Communication between the server and the clients takes place with predefined messages in order to minimize their size and complexity. The messages are C-aligned arrays and their structure is as follows.

Server to client message
flags
global epoch
model variables
...

Client to server message
global epoch
loss
accuracy
model variables / deltas
...

The flags field is used to show to the clients that the model is random or this is the final one and no further communication will be accepted by the server. The GE field is needed in order for the server to know if each received model is addressed for the current GE. If not, the message is ignored. The loss and accuracy fields can be used for more complex algorithms like disregarding local model with inadequate accuracy or greater loss than the previous GE.

At the start of every GE the server sends a message to a every client chosen to participate which is equivalent with an order to start working. When the clients finish their work they respond with one message each, which imply that they finished working. No more communication is needed like confirmation and synchronization messages. In total 2 * N messages are sent every epoch, where N is the number of the clients. As far as the server is concerned, the clients are stateless and every interaction with them is unique. Each message is independent from the rest and must be self sufficient.

Experiment 1

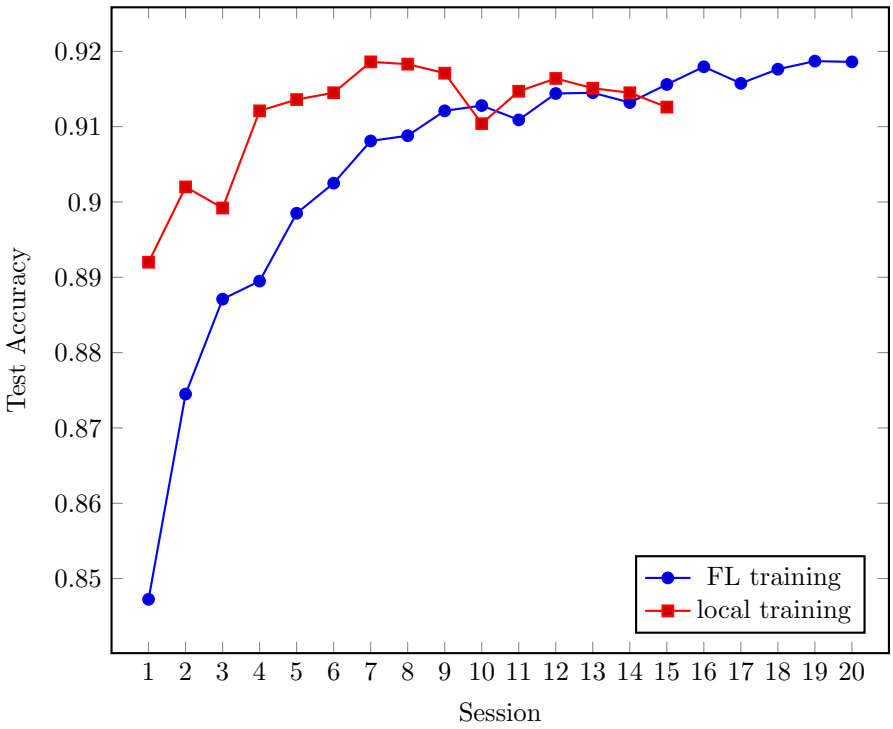
This experiment tests the algorithm with IDD distributed data. A simple CNN that classifies images of the Fashion MNIST dataset is used. The neural network consists of approximately 420,000 Float32 trainable variables and produces messages of 1.7MB. As the server and all the clients are running on the same machine, communication takes place on the operating system's loopback and no substantial timing can be made. The following experiment focus on model accuracy and the convergence rate of the FL algorithm. training dataset which consist of 60,000 images, is split equally between the clients. The following set of parameters and hyperparameters are set.

parameters	
local epochs	1
steps per epoch	3
batch size	10

hyperparameters	
global epochs	500
clients	4

With this parameters and 4 clients, all data will be passed once every 500 global epochs. For the sake of clarity and to not be confused with the global and local epochs, each pass through all data is called a Session.

The accuracy of the FL trained model is compared with the accuracy of a locally trained model. This model has the same batch size, and steps per epoch equal with the number of the data divided by the batch size. Thus, an epoch equals a Session. The following diagram shows the accuracy of the two models on the test data for each Session.



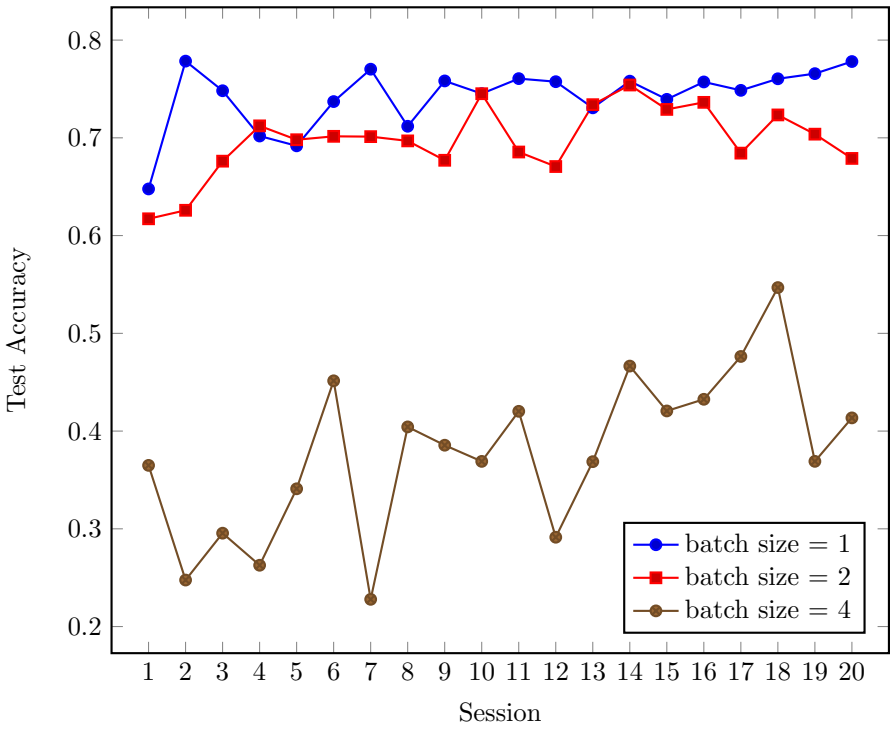
From the above diagram, it is evident that the FL trained model reaches the same accuracy as the locally trained one, albeit on a slower pace. The first model is updated every 150 examples while the second one every 10 examples. According to FL theory, the fewer are the examples that are used every GE, the faster the convergence is. Although, in that case more global epochs are required and the communication overhead increases. In contrast, if more examples are consumed each GE, less communication is required but the convergence rate is smaller and occasionally the model might not even converge. Experiments with different parameters seems to corroborate this.

Another observation from the above diagram is that the FL training combats over-training in some degree. In the locally trained model, each step the training loss was decreasing but this does not correspond to greater test accuracy. In contrast, the averaging of the FL algorithm can incur a rebalancing effect and pull out the local models from local extremum. During training the loss of the client models was not always decreasing.

Experiment 2

In this experiment, the algorithm is tested with heavily unbalanced non-IID data and the previous neural network architecture. The fashion MNIST dataset is used and distributed between five clients. The first client holds all the examples with labels 0 or 1, the second client holds all the examples with labels 2 or 3 etc. This is an extreme case of unbalanced data, clients hold no knowledge of the other classes and if they train a network by themselves, a maximum accuracy of 20% is achieved. In such cases, according to FL theory, better accuracy is achieved by minimizing the batch size and examples used per GE.

For this experiment batch sizes of 1,2 and 4 are used. The local epoch and steps per epoch are set to 1. The global epochs are set in such a way that all data are passed once every session.



From the above diagram it appears that the algorithm follows FL theory. Training with batch size 2 or 4 does not converge, while with batch size 1 accuracy seems to follow a stable positive trend after a few sessions. In addition with reducing batch size, better accuracy can be achieved with operations like data rebalanching or using a large pool of clients.

Future Work

- Modify server to announce the global model to a number of the connected clients instead of all. This can be useful if the server wants to economize how many examples are used each GE in order to have more model updates in each pass through the data.
- In addition of importing a pretrained model, the server could be able to partially train a model locally, before the FL commences.
- Integrate a more complex and larger model to the FL training algorithm.
- Extend the new epoch requirements to consider time passed in the current GE, in order to discard stragglers. The timeout of the event structure can be used. In addition, connections that have not send any data for a few consecutive GE could be dropped.
- Experiment with a large pool of clients, where only a part of them are used every GE.