March 2022 Review

Emmanouil Petrakos

As Machine Learning and Neural Networks become more popular, the demand for additional data is growing. A large amount of data resides in personal devices or servers that are unable to share them due to legal or practical limitations such as privacy laws like GDPR. To mitigate this problem, Federated Learning (FL) algorithms are introduced, in which instead of centralizing data in a server, training gets decentralized by shifting it where the data is stored. The purpose of this work is to identify underlying parallelism in FL training and, if possible, to exploit it. For this to be feasible, the constraints and intricacies of the FL algorithms must first be investigated and defined. Basic assumptions and observations must be reevaluated as most past work focuses on simulations with hundreds of clients, but our goal is to implement FL on hardware with a small number of clients. The experiments that follow are set up in such a way that each one focuses on and isolates a specific aspect of FL.

Server - Client System

The FL training is set up to be cross-device compatible. 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 averaging. This operation continues until the ending requirements are met, e.g. a desired accuracy was reached or a certain amount of time or epochs passed. The system is resistant to random client disconnections and continues operating with the remaining ones.

The server, in addition to orchestrating the training and producing the global model, can import a pretrained model and save the final one. Also, in the first GE, 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 four tasks. Load a pretrained model if it exists, set up a listening socket to receive incoming connections, create a structure to orchestrate the FL training and initialize the environment where the Global Network is evaluated. This structure follows the event driver TCP server paradigm. A single process, handles every event (accept a connection, data available to read or write) 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. 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 is necessary because the size of a message may exceed the operating system's socket windows, resulting in multiple read events per message. 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 examined. Their parameters are the number of local models successfully received, the number of connected clients and the number of clients still working. Provided they are satisfied, a new global model is created and broadcasted to randomly selected clients chosen to take part in the next epoch. 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 created Global Network can be evaluated. Finally, the ending requirements are checked and if they are fulfilled, the final model is saved, send to the clients and the server shuts down.

[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, use a pool of worker threads or just optimize the time consuming sections.

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

Client Overview

At start up, the client needs to complete two tasks. First, set up the training environment. As training/ evaluating is achieved with Tensorflow in Python and the rest of the codebase is developed in C++[3], the Python interpreter must be embedded before the model is compiled. The second task is to create a socket and establish a connection with 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.

The machine learning is modeled using TesnorFlow. The implementation is modular so that different neural network architectures, optimizers, and learning strategies can be used.

Communication Scheme

The server and each client are separate processes with their own memory regions. Data must be shared efficiently and reliably, hence a fitting communication mechanism is required. This is accomplished by using messages with predetermined size and structure in order to reduce the frequency and complexity of the communication. The messages are C-aligned arrays and their structure is as follows.

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

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

The flags field is used to give orders or supplementary information to the clients. These are:

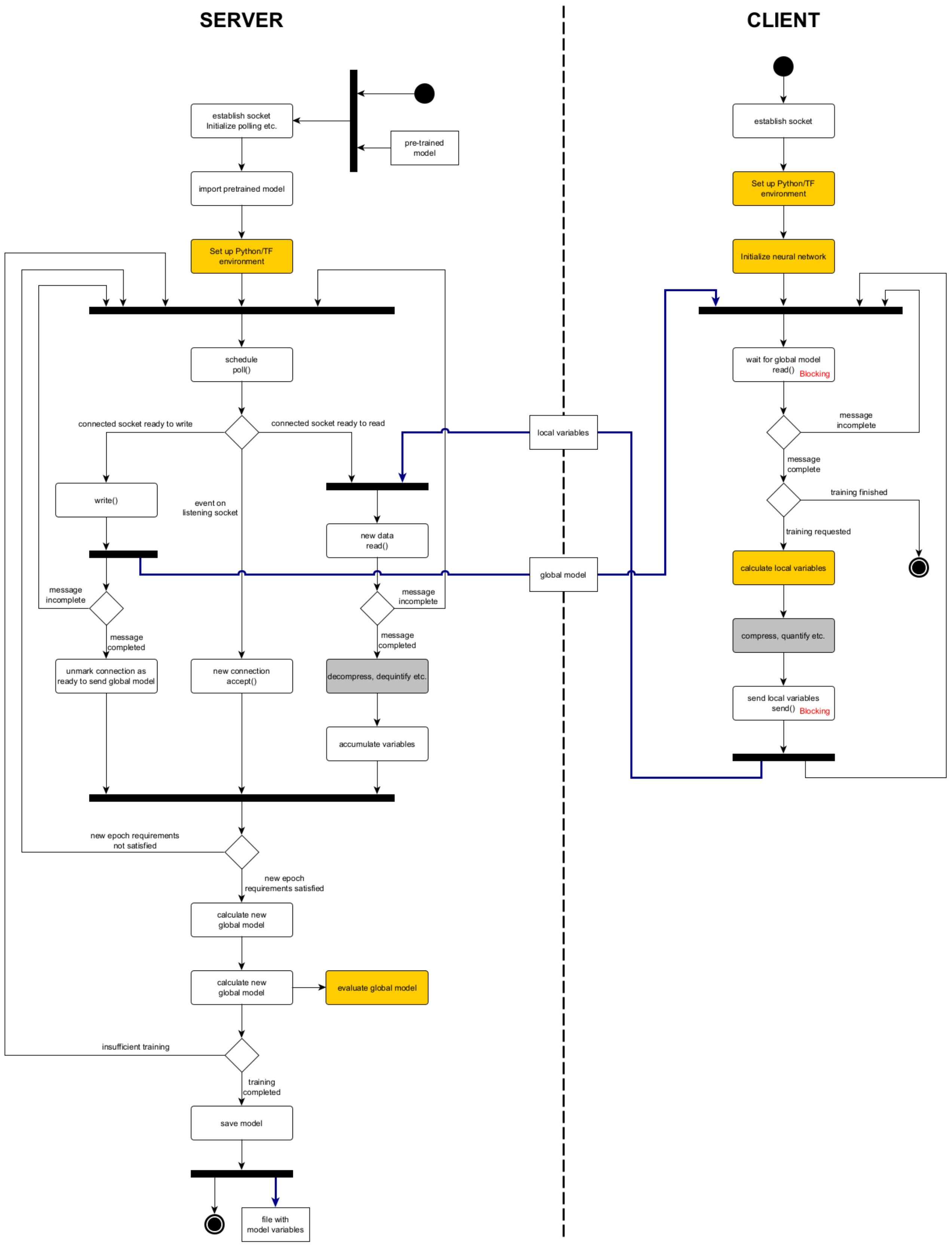
- The received model is random and can be ignored.
- The received model is the final one and no further communication will be accepted by the server.

The GE field is required for the server to determine whether a received model is addressed for the current GE and to accept or ignore it. 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 each GE, the server sends a message to each client chosen to take part, which is essentially an order to begin working. When a client finish its task, they answer with a single message containing the produced local network, indicating that they have completed their work. There is no need for any additional communication, such as confirmation or 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.

[3]The client's architecture is agnostic of the underlying functions and their implementation, i.e. it can perform any federated operation with any implementation as long as its interface is followed.

Server - Client Activity Diagram


SERVER
CLIENT
establish socket Initialize polling etc.
pre-trained model
import pretrained model
Set up Python/TF environment
schedule poll()
connected socket ready to write
connected socket ready to read
write()
event on listening socket
new data read()
message incomplete
message completed
message incomplete
message completed
unmark connection as ready to send global model
new connection accept()
decompress, dequintify etc.
accumulate variables
new epoch requirements not satisfied
new epoch requirements satisfied
calculate new global model
calculate new global model
evaluate global model
insufficient training
training completed
save model
file with model variables
local variables
global model
establish socket
Set up Python/TF environment
Initialize neural network
wait for global model read() Blocking
message incomplete
message complete
training finished
training requested
calculate local variables
compress, quantify etc.
send local variables send() Blocking

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

Experiment 1 - FL with IID data

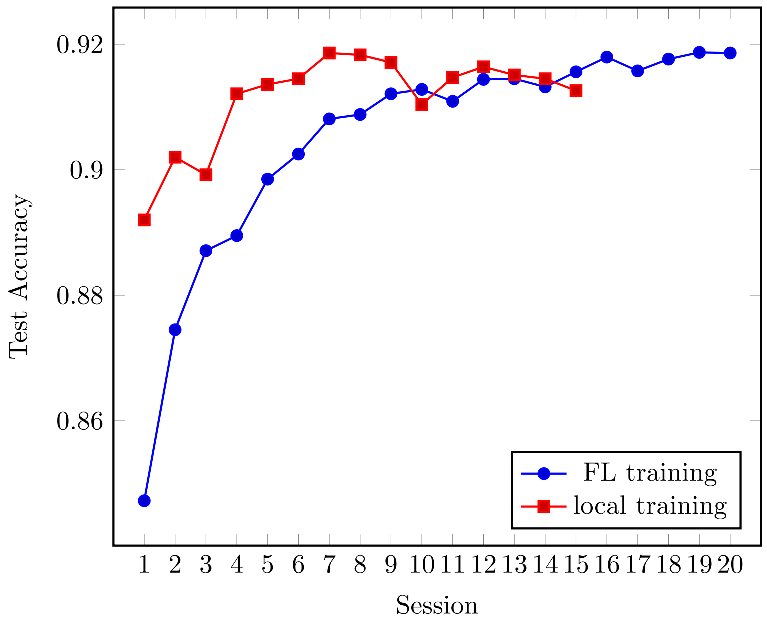
This experiment tests the algorithm with IDD distributed data. A simple CNN that classifies images of the Fashion MNIST dataset and the Adam optimizer are used. The neural network consists of approximately 420,000 Float32 trainable variables which produce messages of 1.7MB. As the server and all the 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 in comparison to locally train a model. The 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 these parameters and 4 clients per GE, all data will be passed once every 500 GEs. 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.



Experiment 1 results

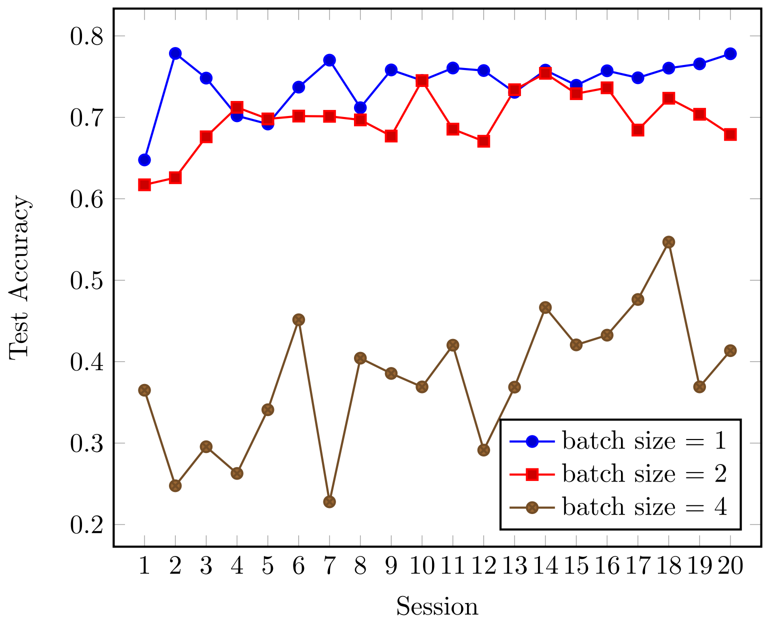
According to the diagram of the experiment, the FL trained model achieves the same accuracy as the locally trained model, albeit at a slower rate. The first model is updated every 150 examples while the second one every 10 examples. According to FL theory, the fewer are the examples that are used every 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. The original FL algorithm without deadline limitations achieved accuracy of 0.92 for Fashion-MNIST with IID data distribution which is comparable with the above results. Experiments with different parameters seems to corroborate this.

Another observation from the above diagram is that FL training combats over-training in some degree. In the locally trained model, the training loss decreased with each step, however this did not translate to higher 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 - FL with non-IID data

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 the 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.



Experiment 2 results

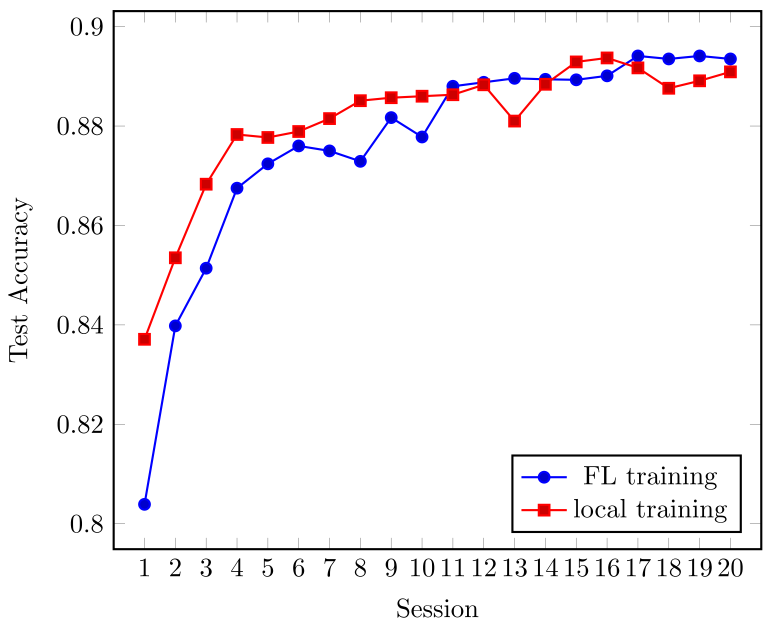
From the above diagram it appears that the algorithm follows FL theory. Training with batch size 2 or 4 does not converge, while with batch size 1 accuracy seems to follow a stable positive trend after a few sessions. In addition with reducing batch size, better accuracy can be achieved with operations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

This experiment should be redone with the advances of the following results and different sets of FL algorithms and advancements.

Experiment 3 - Client Selection

In FL training, it is typical to have more clients available than needed during some epochs. Then, the optimal strategy is to use a subset of them to economize in training examples and keep a consistent rate of data consumption. This functionality is implemented and then tested by integrating the LeNet5 model. Eight clients are connected, but only three randomly selected are used each epoch. Each of owns a shard of the fashion-MNIST dataset, which contains 1/8 of the total number of samples.

Because FL training necessitates multiple runs through the data, the risk of overtraining is increased. To combat this, data reshuffling is introduced. When all of the data in a client possession are consumed, they get reshuffled and new batches are created. This is also used during local training in this experiment. The batch size is set to 20 and the steps per epoch to 2. Additionally, 10000 GE are required to consume the data 20 times during training.



Experiment 3 results

According to the diagram, The algorithm operates normally and reaches similar accuracy with training locally. Also, no signs of overtraining are present.

Experiment 4 - Complex models, greater data per GE consumption

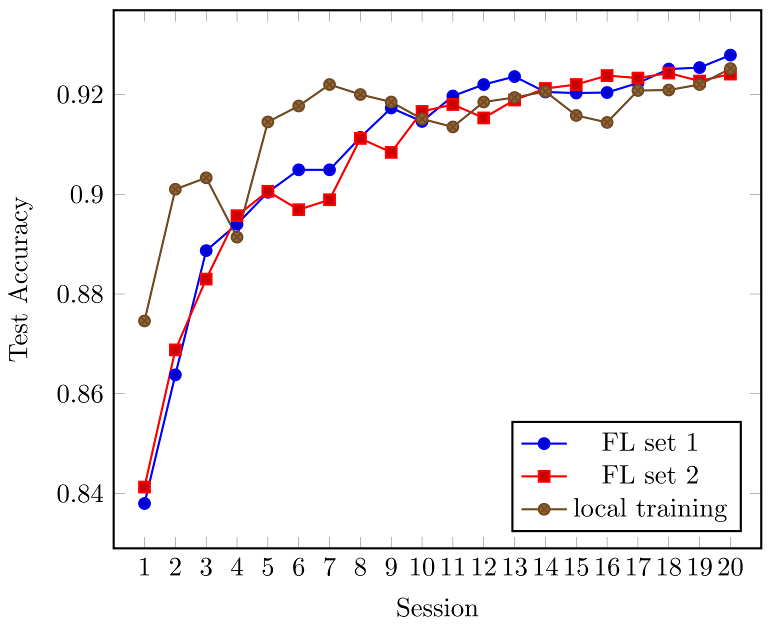
In this experiment, a CNN model architecture is implemented, which utilizes the inception module from GoogLeNet. It consists of 277082 trainable variables and uses the Adam optimizer. Furthermore, the data is IID distributed and data reshuffling is included. The first goal is to assess the response of the FL algorithm with more complex structures.

Two sets of parameters and hyperparameters are used and are detailed in the following array. In both cases, the data is equally divided between 5 clients, but only 3 are used each epoch. The difference between the sets is how many examples are consumed each GE. The second goal of the experiment is to assess the impact of the data utilization per epoch.

parameters	FL set 1	FL set 2	local training
total clients	5	5	1
clients per GE	3	3	1
steps per GE	1	2	examples/batch size
batch size	20	20	20
examples per GE	60	120	all
GEs to use all examples (Session)	1000	500	-

Experiment 4 parameters

The results show that both FL sets reach satisfactory accuracy, albeit at a slower rate than local training. While the second produce slightly less accuracy, it is worth noting that it requires half of the communication of first set. This has the effect of doubling the computation to communication ratio and being a more lucrative target for parallelization. Experiments with larger batch size and steps per epoch should be conducted.



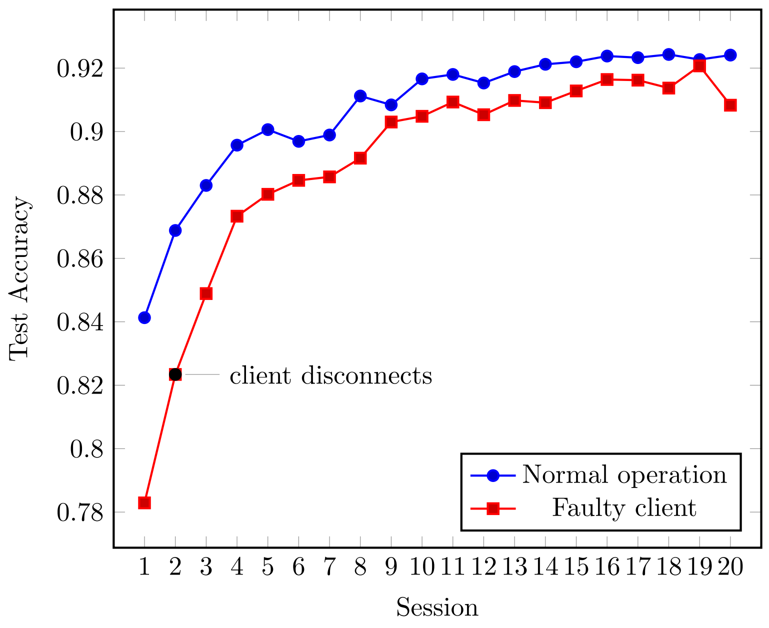
Experiment 4 results

Experiment 5 - Client unreliability

In an edge environment, the clients tend be unreliable and any algorithm must be resilient to random faults. To simulate such a case, the second setting from the previous experiment is used with one extra client, which gets disconnected 1/10 into training. That means for 90% of the training, 1/6 of the data are inaccessible. The CNN from the first experiment is used with the Adam optimizer.

parameters	
total clients	6
clients per GE	3
steps per epoch	1
batch size	20

Experiment 5 parameters



Experiment 5 results

While the accuracy of the model suffers, the effect is not disastrous as training continues and accuracy is still in acceptable range. In a real-world scenario, such a problem can be solved by deferring a portion of the training in a time when new data is available from clients.

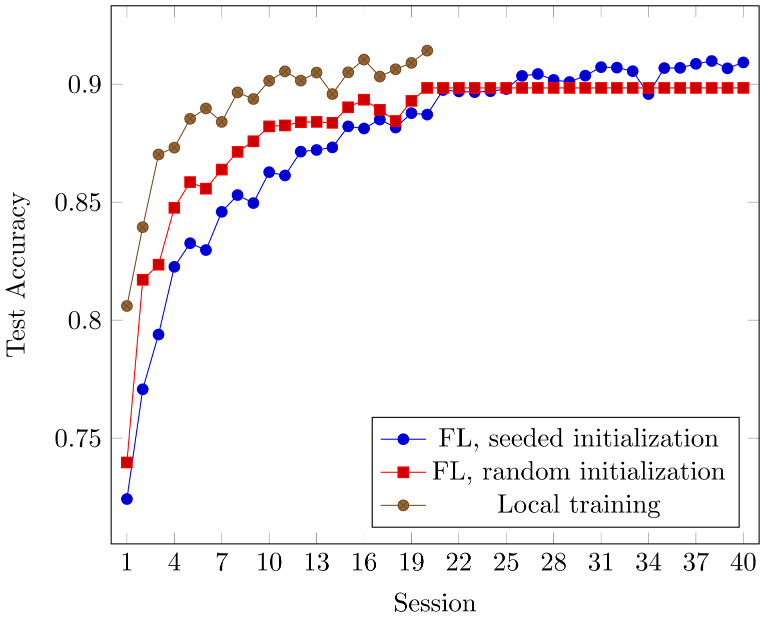
Experiment 6 - NN initialization

The initialization of a neural network used in FL can affect final accuracy and training time. According to previous works, all clients should start with the same initialized network to achieve the best accuracy. The purpose of this test is to evaluate its effect. Furthermore, the SGD optimizer is used.

The CNN architecture from the first experiment is used with the Glorot initializer, which is seeded in order to produce the same variables on all clients. The model is compared with a locally trained one and a FL trained one without seeded initialization.

parameters	FL, seeded init	FL, random init	local training
total clients	5	5	1
clients per GE	3	3	1
steps per GE	5	5	examples/batch size
batch size	20	20	20
examples per GE	300	300	all
GEs to use all examples (Session)	200	200	-

Experiment 6 parameters



Experiment 6 results

The unseeded network seems to reach a local minima and its accuracy plateaus. In contrast, the seeded one continues improving till it achieves the accuracy of the locally trained model.

Experiment 7 - learning rate (LR) decay strategies

Another aspect of FL worth investigating is learning rate (LR) decay strategies. Three of them are implemented.

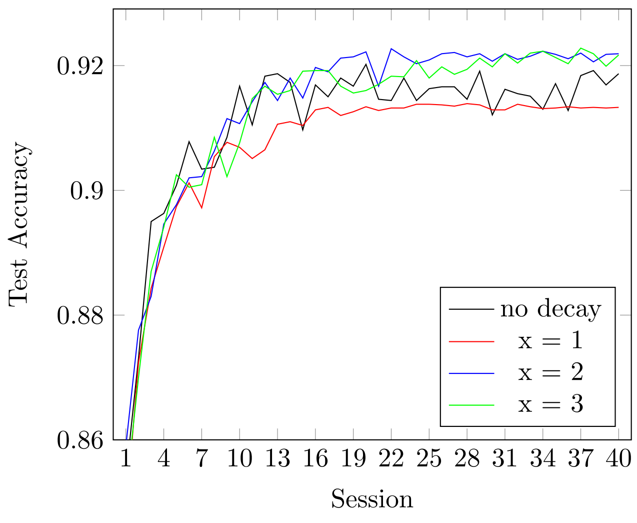
- Decay the LR every set number of GEs. All clients have the same LR at every moment.
- Decay the LR of a client based on the number of participated GEs. If a subset of the clients is used every GE, some clients may have been selected more times than others and as a result they will have lower LR.
- The final strategy is to reduce a client's LR each time its data is completely utilized. This is an extension of the second strategy, where instead of decaying slowly the LR every few rounds, there is a big drop every $\frac{\sum clients\ data}{\sum clients\ data\ used\ per\ GE}$ rounds of training.

Each strategy is tested three times with different decay periods. The decay period dictates how often the decay applies. E.g. the second strategy with the a decay period of three means that LR decays every three participated rounds. The results are compared with a FL trained model without LR decay.

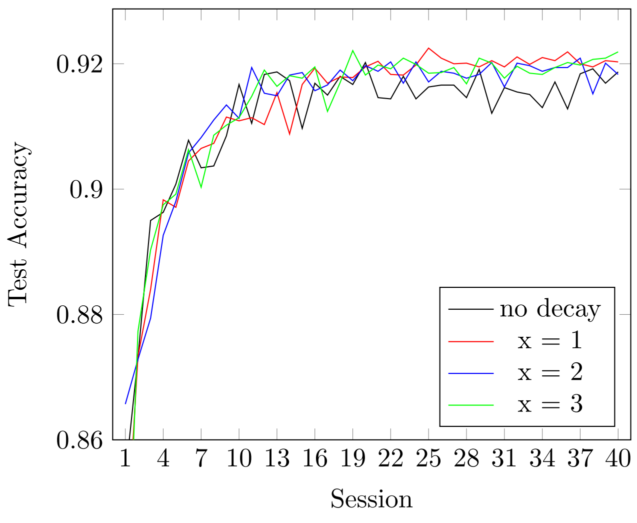
parameters	FL, no decay	FL, strategy 1	FL, strategy 2	FL, strategy 3
total clients	5	5	5	5
clients per GE	3	3	3	3
steps per GE	5	5	5	5
batch size	20	20	20	20
initial LR	1e-2	1e-2	1e-2	1e-2
LR decay	-	0.999	0.999	$\frac{0.999 * \sum clients\ data}{\sum clients\ data\ used\ per\ GE}$
decay interval (x = decay period)	-	x GEs	x participated GEs	$\frac{x\ participated\ GEs * \sum clients\ data}{\sum clients\ data\ used\ per\ GE}$

Experiment 7 parameters

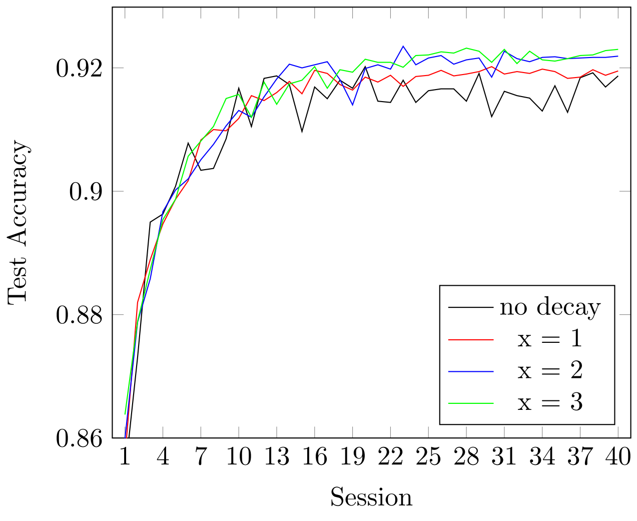
The following tables compare each strategy against the no LR decay FedSGD respectively.



Strategy 1



Strategy 2



Strategy 3

All strategies seems to to perform slightly better than the baseline, except the first one with decay period = 1. In that case, the decay is too fast and the LR degenerates in a state that cannot substantially alter the weights of the NN. The last strategy appears to be the most promising, whereas outperforming the others is the most straightforward.

Experiment 8 - FederatedAveraging (FedAvg)

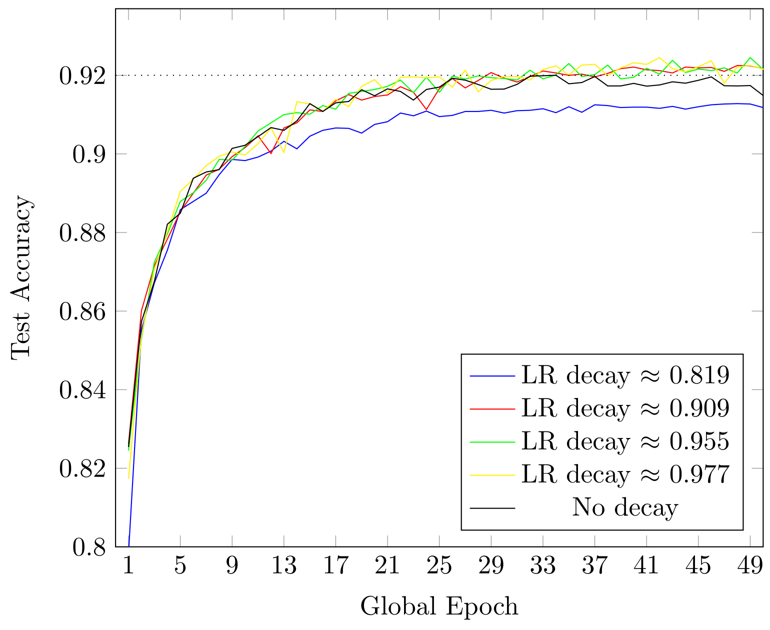
FedAvg is a popular FL algorithm in which clients use all of their data and perform multiple SGD iterations during a participating GE. As a result, few rounds of communication are required, and the parameters should be tuned accordingly. In the prior experiment, a client had to participate in 200 GEs to consume all of its data, whereas with FedAvg, only one GE (at most) is required. As a result, the LR decay must be calibrated in consideration of the fewer decay events. This is the subject of this experiment.

In the prior experiment, with the second or the third strategy a LR of 0.818649 is reached when the examples of a client are exhausted. For the first test, this is the value of the LR decay. Furthermore, three more test are performed, each with half the descent slope of the previous one. Finally, a test without LR decay is run.

parameters	FedAvg
total clients	5
clients per GE	3
local epochs	1
steps per epoch	600
batch size	20
initial LR	1e-2

Experiment 8 parameters

From now on, the y axis of the tests is the elapsed GEs, as in each GE all data are consumed. Furthermore two metrics will be monitored, maximum accuracy and communication rounds to reach a target accuracy of 0.92.



LR decay	Max accuracy	0.92 @GE
0.819	0.913	-
0.909	0.9232	30
0.955	0.9245	32
0.977	0.9245	27
No decay	0.9224	34

Experiment 8 results

A number of observations can be made. To begin with, much fewer communication rounds are required to achieve the desired accuracy in comparison with the previous experiments.[4] However, there is a hidden cost in that less averaging occurs and its rebalancing effect is reduced. When comparing the different LR decay values, it seems that the most conservative options perform better; decaying the LR too quickly causes the NN to set in a low accuracy value.

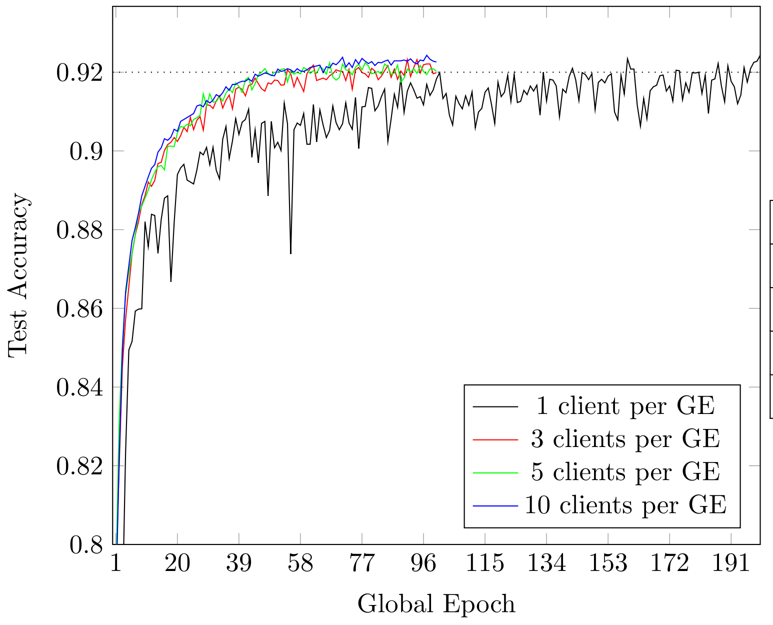
[4]Comparing with the experiment 7, x100-200 less communication with the same computation cost.

Experiment 9 - Client Participation / Increasing parallelism

This experiment explores the amount of multi-client parallelism that can be exploited and its effect on training. The dataset is split between 10 clients, each one holding 6000 training examples. The CNN architecture from the previous experiment is trained multiple times with different number of participating clients per GE. Training with one client per epoch serves as a baseline. The relative reduction in communication is calculated for all other tests.

Test	A	B	C	D
total clients	10	10	10	10
clients per GE	1	3	5	10
local epochs	1	1	1	1
steps per epoch	300	300	300	300
batch size	20	20	20	20
initial LR	1e-2	1e-2	1e-2	1e-2
LR decay	0.977	0.977	0.977	0.977

Experiment 9 parameters



client per GE	0.92 @GE
1	159
3	61 (2.6×)
5	46 (3.4×)
10	51 (3.1×)

Experiment 9 results

Using more clients every GE substantially lowers the rounds of communication needed to achieve the target accuracy. Others works that are focused on simulations of hundreds of clients with small dataset each, have shown similar reductions. Empirically, using a bit more than half of the clients in each GE appears to yield the best results. If all the clients are used every GE, in some cases like non-IID data distribution, the network may not converge in an acceptable solution.

Experiment 10 - Increasing computation per client

In this section, the computation per client is fluctuated in order to investigate its effect on the number of required GEs. The number of local SGD updates per GE can be increased by either executing more local epochs (LE), decreasing the batch size (B), or both. The CNN from the previous experiment is used, the dataset is split between 5 clients and 3 of them participate each epoch. Learning rate and its decay are amortised in order to have the same behavior across all tests. The target is to minimize the number of required GEs to reach the target accuracy of 0.92.

Local epochs = 1		
B	updates/GE	0.92 @GE
600	20	309
300	40	212
100	120	72
80	150	54
40	300	31
20	600	25
10	1200	18

Local epochs = 3		
B	updates/GE	0.92 @GE
600	60	-
300	120	67
100	360	36
80	450	25
40	900	13
20	1800	7
10	3600	15

Experiment 10 results

According to the experimental results, increasing local updates directly decreases the required global updates. Unlike most works, this one concentrates on small groups of clients with large dataset in each. As a result, increasing the number of local epochs produces inconsistent results due to the introduction of overfitting on the local models. On the other hand, providing that the batch size is large enough to completely utilize the client's hardware parallelism, there is no cost in lowering it.

Future Work

- Experiment with dropout layers. Through the last techniques, the number of GEs has been reduced and the rebalancing effect of FL training minimized. Another way to combat overfitting should be introduced, but the effect of dropout layers isn't immediately obvious; in every participating client different variables are dropped.
- Test CE-FedAvg, a variation of FedAvg using the Adam optimizer.
- Quantizing the NN variables from Float32 to Int8 after local training, can reduce the network communication by 3/4. Test its effect on accuracy and convergence.
- Maybe reevaluate non-IID data performance with the advances from the other experiments. Repeating experiment 10 with non-IID distribution should be enough.
- Experiment with transfer learning; this might be getting out of scope of this work.