A Performance Evaluation of Federated Learning Algorithms

Adrian Nilsson[*][†]
Fraunhofer-Chalmers Centre
Gothenburg, Sweden
adrian.nilsson@fcc.chalmers.com

Simon Smith[*][†]
Fraunhofer-Chalmers Centre
Gothenburg, Sweden
simon.smith@fcc.chalmers.com

Gregor Ulm[†]
Fraunhofer-Chalmers Centre
Gothenburg, Sweden
gregor.ulm@fcc.chalmers.se

Emil Gustavsson[†]
Fraunhofer-Chalmers Centre
Gothenburg, Sweden
emil.gustavsson@fcc.chalmers.se

Mats Jirstrand[†]
Fraunhofer-Chalmers Centre
Gothenburg, Sweden
mats.jirstrand@fcc.chalmers.se

ABSTRACT

Federated learning is an approach to distributed machine learning where a global model is learned by aggregating models that have been trained locally on data-generating clients. Contrary to centralized optimization, clients can be very large in number and face challenges of data and network heterogeneity. Examples of clients include smartphones and connected vehicles, which highlights the practical relevance of federated learning. We benchmark three federated learning algorithms and compare their performance against a centralized approach where data resides on the server. The algorithms Federated Averaging (FedAvg), Federated Stochastic Variance Reduced Gradient, and CO-OP are evaluated on the MNIST dataset, using both i.i.d. and non-i.i.d. partitionings of the data. Our results show that FedAvg achieves the highest accuracy among the federated algorithms, regardless of how data was partitioned. Our comparison between FedAvg and centralized learning shows that they are practically equivalent when i.i.d. data is used. However, the centralized approach outperforms FedAvg with non-i.i.d. data.

CCS CONCEPTS

• **Computing methodologies → Machine learning approaches**;
• **Computer systems organization** → *Client-server architectures*;

KEYWORDS

Machine learning, Federated learning, Algorithm evaluation

ACM Reference Format:
Adrian Nilsson, Simon Smith, Gregor Ulm, Emil Gustavsson, and Mats Jirstrand. 2018. A Performance Evaluation of Federated Learning Algorithms. In *Second Workshop on Distributed Infrastructures for Deep Learning (DIDL '18), December 10–11, 2018, Rennes, France.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3286490.3286559

1 INTRODUCTION

In the automotive industry, analysis of user data from a fleet of vehicles may grant insights into user needs as well as vehicle behavior and driving environments. Current approaches to vehicular data analysis involve sending compressed sensor-data from all vehicles to a central server that carries out data analysis tasks [9]. However, a modern car can produce hundreds of gigabytes of data per day [7], which means that data transfer and central data storage are infeasible in practice. Furthermore, user data is private in nature, which raises privacy concerns about transferring and storing such data on a central server.

With a recent approach called *federated learning* [13], only a model, e.g. parameters of an Artificial Neural Network (ANN), has to be communicated between the server and client devices. This can reduce the amount of required data transfer while also alleviating privacy concerns since all computations are performed locally on clients, using their own data. Thus, a computation that was previously performed on a powerful server has been distributed to many less powerful devices such as mobile phones or on-board units in vehicles.

Federated learning is different from other distributed data-parallel problems (e.g. data center distribution) in several aspects. While both approaches aim to optimize their learning objective, federated learning algorithms have to account for the fact that communication with edge devices takes place over unreliable networks with very limited upload speeds. Since communication in federated learning is much more expensive than computation, it is crucial to minimize communication. Therefore, we define performance as the highest classification accuracy achieved after a given amount of communication. The communication can either be in terms of communication rounds between a server and its clients, or uploaded models from each client.

Another fundamental difference between existing distributed machine learning approaches and federated learning is that earlier approaches make underlying assumptions about the training data that are too strong in the federated setting. Distributed optimization algorithms typically assume that [10, 13]:

- Data is evenly distributed across clients
- Client-side data are *independent and identically distributed* (i.i.d.) samples from the overall distribution
- The number of clients is much smaller than the average number of locally available training examples per client

[*] Both authors contributed equally to the paper.
[†] Also with Fraunhofer Center for Machine Learning.

DIDL '18, December 10–11, 2018, Rennes, France

ACM ISBN 978-1-4503-6119-4/18/12.
https://doi.org/10.1145/3286490.3286559

In contrast, federated optimization algorithms *cannot* make these assumptions.

Our work focuses on classification tasks using ANNs in the setting of federated learning. We refer to this simply as *federated optimization*, although this is somewhat imprecise since federated learning is not inherently constrained to optimizing ANN models. Three of the few general federated optimization algorithms proposed in the literature are Federate Averaging (FedAvg) [13], Federated Stochastic Variance Reduced Gradient (FSVRG) [10], and CO-OP [14], which are the algorithms we consider in this paper. Our contributions are (1) a performance comparison of three federated learning algorithms, and (2) comparing them to regular centralized learning.

2 BACKGROUND

This section motivates federated learning from a big data perspective and briefly reviews the underlying optimization problem of empirical risk minimization.

2.1 Big data challenges

Rich sources of data are found in the mobile devices we use everyday, be it phones, tablets, or automotive vehicles. These devices are equipped with a plethora of different sensors capable of producing vast amounts of data. For example, a moderately sized fleet of 1,000 test vehicles is estimated to be able to produce data in the order of terabytes each day [9]. Also, Hitachi [7] claims that a modern internet-connected vehicle produces data in excess of 25GB/hour. This data abundant setting is often referred to as *big data*.

Centralized data analysis has limitations in a big data setting. Notably, if data is generated on mobile edge devices, wireless data transfer over cellular networks becomes a bottleneck. Although the LTE and LTE-Advanced standard of 4G specifies peak upload rates of 50Mb/s and 500Mb/s respectively [12], a case study on data rates in central South Korea [8] shows that practical performance is far from ideal even in an urban environment. The case study shows no improvements from LTE to LTE-Advanced in terms of upload bandwidth, were the best average upload bandwidth was approximately 13.5Mb/s. Centralization of vehicle data is infeasible with such data rates, which would be even lower in rural areas.

2.2 Motivation for federated learning

Distributed data center optimization typically requires control over the data distribution since these approaches rely on balanced and i.i.d. data assumptions. However, these assumptions are too strong when performing learning tasks on a heterogeneous ecosystem of edge devices.

Instead of processing data centrally, federated learning proposes to have a set of edge devices perform learning tasks locally and only communicate an updated model to a coordinating server. More specifically, a server learns a shared global model by aggregating locally trained models from a possibly very large number of clients. Furthermore, these clients typically have unbalanced and non-i.i.d. data as well as limited data transfer capabilities.

The federated approach also has advantages over the centralized alternative when it comes to data privacy. For instance, federated learning applies the General Data Protection Regulation's (GDPR) data minimization principle [5] since only the learned model, and no raw data, is processed centrally. Communicated models are also temporary in the sense that they are immediately discarded after being merged into the global model, which is an application of the storage and purpose limitation principles of GDPR.

2.3 The optimization problem

The algorithms we consider optimize the finite-sum objective

$$\min_{w \in \mathbb{R}^d} f(w) , \qquad f(w) := \frac{1}{n} \sum_{i=1}^{n} f_i(w), \tag{1}$$

where w is a vector that contains d model parameters. In supervised learning, we treat the function $f_i(w)$ as a loss function $f_i(w) = \ell(x_i, y_i; w)$, where an input-output pair (x_i, y_i) is one of n given labeled examples, often referred to as *training examples*. The objective function $f(w)$ is defined by the model parameters w conditioned on n labeled examples. The problem can thus be interpreted as finding the w which minimizes the average loss over all n training examples.

In a big data context, where the number of training examples is too large to be stored on one computer, we must distribute the computation to multiple computers. If the number of training examples held by client k is denoted by $n_k = |\mathcal{P}_k|$, then we can rewrite the objective function as a weighted sum over all $F_k(w)$:

$$f(w) = \sum_{k=1}^{K} \frac{n_k}{n} F_k(w) , \quad \text{where} \quad F_k(w) := \frac{1}{n_k} \sum_{i \in \mathcal{P}_k} f_i(w). \tag{2}$$

Distributing the data and computational burden leads us to reformulate the objective function $f(w)$ from Eq. (1) to (2). Assume that there are K clients arcoss which data and computation are distributed. Each client then holds a part $\mathcal{P}_k$ of all training examples, and computes $F_k(w)$, which is the average loss on client k.

3 ALGORITHMS

Our evaluation includes the synchronous FedAvg and FSVRG algorithms as well as the asynchronous CO-OP algorithm. This section describes these algorithms and gives their psuedocode.

3.1 Federated Averaging (FedAvg)

FedAvg orchestrates training via a central server which hosts the shared global model w_t, where t is the communication round. However, the actual optimization is done locally on clients using, for instance, Stochastic Gradient Decent (SGD). FedAvg has five hyperparameters: the fraction of clients C to select for training, the local mini-batch size B, the number of local epochs E, a learning rate η, and possibly a learning rate decay λ. The parameters B, E, η, and λ are commonly used when training with SGD. However, here E stands for the total number of iterations through the local data *before* the global model is updated.

The algorithm starts by randomly initializing the global model w_0. One communication round of FedAvg then consists of the following: The server selects a subset of clients S_t, $|S_t| = C \cdot K \geq 1$, and distributes the current global model w_t to all clients in S_t. After updating their local models w_t^k to the shared model, $w_t^k \leftarrow w_t$, each client partitions its local data into batches of size B and performs

E epochs of SGD. Finally, clients upload their trained local models w^k_{t+1} to the server, which then generates the new global model w_{t+1} by computing a weighted sum of all received local models. The weighting scheme is dependent on the number of local training examples, as described in Algorithm 1 on line 7.

```
Algorithm 1: FederatedAveraging
1 initialize $w_0$
2 for each round $t = 0, 1, \ldots$ do
3     $m \leftarrow \max(\lfloor C \cdot K \rfloor, 1)$
4     $S_t$ = random set of $m$ clients
5     for each client $k \in S_t$ in parallel do
6         $w^k_{t+1}$ = ClientUpdate($k, w_t$)
7     $w_{t+1} = \sum_{k \in S_t} \frac{n_k}{n_\sigma} w^k_{t+1}$,    $n_\sigma = \sum_{k \in S_t} n_k$
```

3.2 Federated Stochastic Variance Reduced Gradient (FSVRG)

The idea behind FSVRG is to perform one expensive full gradient computation centrally, followed by many distributed stochastic updates on each client. A stochastic update is performed by iterating through a random permutation of the local data, performing one update per data point. Standard FSVRG only has one hyperparameter: the stepsize h. However, this stepsize is not used directly. Instead, client k has a local stepsize h_k that is inversely proportional to n_k, i.e. $h_k = h/n_k$. The motivation behind h_k is that clients should make roughly the same amount of progress when n_k varies greatly from client to client [10, p.22].

Algorithm 2 gives a complete description of FSVRG, where one iteration is performed as follows: First, to compute a full gradient, all clients download the current model w_t and compute loss gradients with respect to their local data. Clients then upload their gradients, which the server aggregates to form the full gradient $\nabla f(w_t)$. Next, all clients download $\nabla f(w_t)$ and initialize their local model w^k_t and local step-size h_k. After creating a random permutation of their local data, clients will iteratively perform n_k SVRG updates using both the local and the full gradient as well as a client specific stepsize h_k. Finally, when all clients have computed and uploaded their final w^k_{t+1}, the server combines all w^k_{t+1} to form a new global model w_{t+1}, similar to FedAvg.

FSVRG in its original form [10, Alg. 4] is primarily concerned with sparse data in the sense that some features are seldom represented in the dataset, or are only present on few clients. This sparsity structure is exploited by multiplying gradients and model parameters with diagonal matrices that contain information about how frequently features are represented. However, this scaling is only possible because the dimension of the model is the same as the dimension of the input in the Support Vector Machine (SVM) model they consider. In a neural network model, however, the number of parameters is generally much larger than the input dimension. Since we cannot apply these scaling matrices, they are excluded from Algorithm 2.

```
Algorithm 2: Federated SVRG
initialize $w_0$
$h \leftarrow$ stepsize
$\{\mathcal{P}_k\}_{k=1}^K$ = data partition
for each round $t = 0, 1, \ldots$ do
    Compute $\nabla f(w_t) = \frac{1}{n} \sum_{i=1}^n \nabla f_i(w_t)$
    for all $K$ clients in parallel do
        initialize: $w^k_{t+1} \leftarrow w_t$, and $h_k = \frac{h}{n_k}$
        let $\{i_s\}_{s=1}^{n_k}$ be a permutation of $\mathcal{P}_k$
        for $s = 1, \ldots, n_k$ do
            $\Theta \leftarrow \nabla f_{i_s}(w^k_{t+1}) - \nabla f_{i_s}(w_t) + \nabla f(w_t)$
            $w^k_{t+1} \leftarrow w^k_{t+1} - h_k \cdot \Theta$
    $w_{t+1} = \sum_{k=1}^K \frac{n_k}{n} w^k_{t+1}$    // update global model
```

3.3 CO-OP

Whereas FedAvg and FSVRG rely on synchronized model updates, CO-OP [14] proposes an asynchronous approach. This approach will immediately merge any received client model with the global model. Each client k has an age a_k associated with its model and the global model has age a. The model age difference, $a - a_k$, is used to compute a weight when merging models. This is motivated by the fact that in an asynchronous framework, some clients will train on outdated models while others will train on more up-to-date models.

A local model will only be merged if $b_l \leq a - a_k \leq b_u$, for some choice of integers $b_l < b_u$. The intuition behind this acceptance rule is that we neither want to merge outdated models ($a - a_k > b_u$) nor models from overactive clients ($a - a_k < b_l$). The lower and upper bounds, b_l and b_u, can therefore be thought of as an *age filter*. CO-OP also inherits all hyperparameters from its underlying optimization algorithm, for instance SGD.

The training procedure is as follows: Each client has its own training data, and performs E rounds of an optimization algorithm before requesting the current global model age a from the server. The client now decides whether or not its age difference meets the restrictions. Should the local model be outdated, the client reconciles with the global model and starts over. Should the client instead be overactive, it just continues training. Otherwise, the local model is uploaded to the server for merging. The pseudocode of CO-OP is presented in Algorithm 3.

There is one major difference in how the training data is accessed in Algorithm 3 compared to the original description of CO-OP. We let a client have access to all its data at all times, while Wang [14] aggregated data into a batch at certain randomized time intervals. In Wang's method, a client could therefore only access one batch at a time. This means that our implementation trained on more data and in the same way as our implementations of the other algorithms.

3.3.1 *Restrictions on the age filter.* CO-OP introduces two additional parameters, namely b_l and b_u, but little guidance is provided

Algorithm 3: CO-OP

```
$w = w_1 = \ldots = w_K \leftarrow w_0$
$a \leftarrow b_l$
$a_1 = \ldots = a_K \leftarrow 0$
// Each client $k$ independently runs:
while $true$ do
    $w_k \leftarrow$ ClientUpdate($w_k$)
    Request and receive the model age $a$ from the server
    if $a - a_k > b_u$ then
        // Client is outdated
        Fetch $w, a$ from the server
        $w_k \leftarrow w$, $a_k \leftarrow a$
    else if $a - a_k < b_l$ then
        continue // Client is overactive
    else
        // Normal update
        $w_k, a_k \leftarrow$ UpdateServer($w_k, a_k$) = {
                $w \leftarrow (1-\alpha)\cdot w + \alpha \cdot w_k$ ,    $\alpha \leftarrow (a - a_k + 1)^{-\frac{1}{2}}$
                $a \leftarrow a + 1$
                return $w, a$
            }
```

as to how one should choose these values. Only the intuitive constraint $b_l < b_u$ is given in the original paper [14]. However, choosing these parameters arbitrarily with only this constraint in mind may cause a deadlock.

If all clients are deemed overactive, the algorithm deadlocks. We identify two additional constraints that should be fulfilled to avoid this deadlock: $b_l < K$ and $b_u \geq 2b_l$. If the first constraint is unfulfilled, CO-OP is guaranteed to deadlock after K updates. This follows from the intuition of b_l; at least b_l normal updates must be performed by distinct clients before a client is allowed another normal update. The second constraint says that a deadlock might occur if the difference between b_l and b_u is too small. A proof of the second constraint is given in Appendix A.4.

4 BENCHMARK DESIGN

This section intends to make our benchmark design and methodology explicit. That includes how we ran benchmarks, how we evaluated the results, how training data were partitioned, and for how long the algorithms trained.

4.1 Benchmarking

All our distributed benchmarks contained a total of 100 clients, which were distributed as evenly as possible to our three machines (see Appendix A.2), where one machine also acted as the server.

CO-OP is affected by how clients are distributed to the machines in our experimental setup. Clients simulated on the same machine that runs the server will be faster to upload since they do not have to communicate over Ethernet. Therefore, these clients are often deemed overactive by the CO-OP age filter, which in turn may cause clients on other machines to be outdated. To ensure consistent behavior in all CO-OP benchmarks, the machines always had the same simulated clients.

For FedAvg and FSVRG we evaluated the global model on a test set once per communication round. In CO-OP, however, evaluating the global model every time it is updated is not as meaningful since an update from a single client makes little progress compared to a synchronous communication round. Therefore, we used an evaluation frequency of once every 10th update for CO-OP. This is comparable to FedAvg with $C = 0.1$ in the number of models uploaded per evaluation.

4.2 Data distribution

We use the MNIST dataset of handwritten digits [11] in our benchmarks. The MNIST training data must also be divided among our clients. A straightforward way is to shuffle the data and distribute an equal amount to all clients. This gives us i.i.d. data; no client is special. Assuming i.i.d. data in a federated setting does not, however, reflect a realistic scenario. Therefore, we also use non-i.i.d. data.

We use the term *non-i.i.d. data* to refer to samples that have been purposely given to a client because they have certain values. We did this by sorting and dividing the data into *shards*, following the approach of [13]. The step-by-step process is to sort the data, divide the data into equally sized shards, and then randomly assign a number of these shards to each client. In the case of MNIST with 100 clients, each client had two shards of size 300. An example of how our i.i.d. and non-i.i.d. distributions differ from each other is given in Fig. 1.

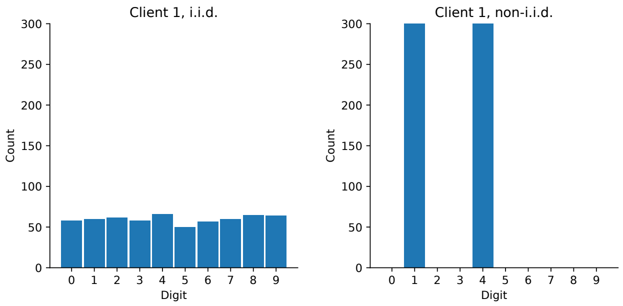


Figure 1: Side-by-side comparison of an i.i.d. and non-i.i.d. distribution taken from one of our clients.

We also create i.i.d. and non-i.i.d. data partitionings for 5-fold cross-validation. The i.i.d. variant is created by shuffling and dividing the full MNIST dataset, including the test set, into five folds. Each fold is then partitioned into 100 equally sized subsets, one for each client. Since MNIST consists of 70,000 labeled images, each client will have 140 unique samples in each fold, 560 in total if one fold is left out.

The process to create the non-i.i.d. partitioning with folds is as follows:

(1) Shuffle the dataset
(2) Divide it into five folds
(3) Sort each fold

(4) Divide each fold into 200 shards of size 70
(5) Uniquely assign two random shard *positions* to each client
(6) Assign two shards to each client at its shard positions from each fold

A difference from the previous non-i.i.d. approach is the decreased shard size from 300 to 70, but each client also receives five times more shards, one from each fold. However, this does not mean that each client is expected to have five times as many unique labels. Shards taken from the same position in each fold are expected to contain the same labels because each fold is sorted and no fold is expected to have more samples of a particular label than another. This is the reason why this distribution is considered to be non-i.i.d. In practice, most clients had two unique labels, as expected.

4.3 Evaluation approach

We follow the Bayesian approach recommended by [1] for our algorithm performance comparison. The Bayesian correlated t-test [4] is used since we consider performance on a single dataset (MNIST). Each test results in a posterior distribution that describes the mean difference in accuracy between the tested classifiers. Since the posterior is a probability density function, it can be integrated to infer a probability of the hypothesis (the mean difference in accuracy) given the observed data. The input to the Bayesian t-test[1] is a list of differences in accuracy of two classifiers from i iterations of k-fold cross-validation. For us, this motivates a distributed partitioning of the MNIST data into folds, as described in Sec. 4.2.

As an example of how to interpret the results from a Bayesian correlated t-test, consider the posterior shown in Fig. 2. The tested classifiers, A and B, were evaluated using three runs of 5-fold cross-validation, after which classifier A had an average accuracy of 93.5% and classifier B 92.9%. The area under this distribution in the interval $(-\infty, -0.01)$ is 0.379, which represents the probability that classifier A is practically better than classifier B. Similarly, the area in the interval $(0.01, \infty)$ equals 0.055 and represents the probability that classifier B is practically better than classifier A. The area between $[-0.01, 0.01]$, the defined region of practical equivalence (rope), is 0.566, which we again interpret as a probability.

One can argue that the two classifiers compared in Fig. 2 are practically equivalent because a majority of the distribution's area is within the rope. This conclusion is nevertheless uncertain since 43.4% of the area is outside the rope. If the distribution instead had, say, 95% of its area to the left of −0.01, it may make sense to state that A is practically better than B. However, we could also argue that A should be the preferred algorithm since the probability that B is practically better than A is only 5.5%.

4.4 Termination criterion for cross-validation

Running the algorithms until they converge would take too much time on our experimental setup. Instead, we need a fair measurement that works for all of them, such as the same total number of uploaded models from clients. Network throughput is important in federated learning and the upload speed is especially important since it is generally slower than the download speed. Therefore, we used a termination criterion of 100 estimated uploads per client, which is equivalent to 10,000 uploads from 100 clients for FedAvg ($C = 0.1$), CO-OP, and FSVRG. In terms of communication rounds, this translates to 1000 communication rounds for FedAvg and 50 for FSVRG. FSVRG is only allowed 50 communication rounds because each client will upload two times each round; one upload for the full gradient and one for the updated model.

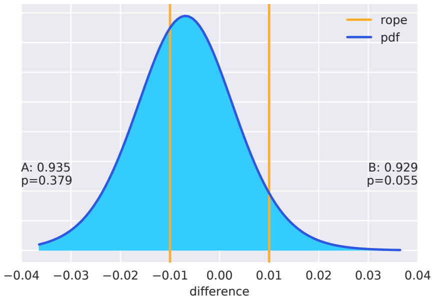


Figure 2: Posterior probability distribution of a correlated Bayesian t-test between classifiers *A* and *B*. The vertical lines defines a region of practical equivalence where the mean difference in accuracy is no more than ±1%.

5 EXPERIMENTAL RESULTS

This section contains comparisons between FedAvg, CO-OP, FSVRG, and centralized learning where all data resides on a server. We use a multi-layer perceptron (see Appendix A.1) to evaluate them via cross-validation on MNIST. Final comparisons are made using Bayesian correlated t-tests. However, we first need good hyperparameter values for each algorithm.

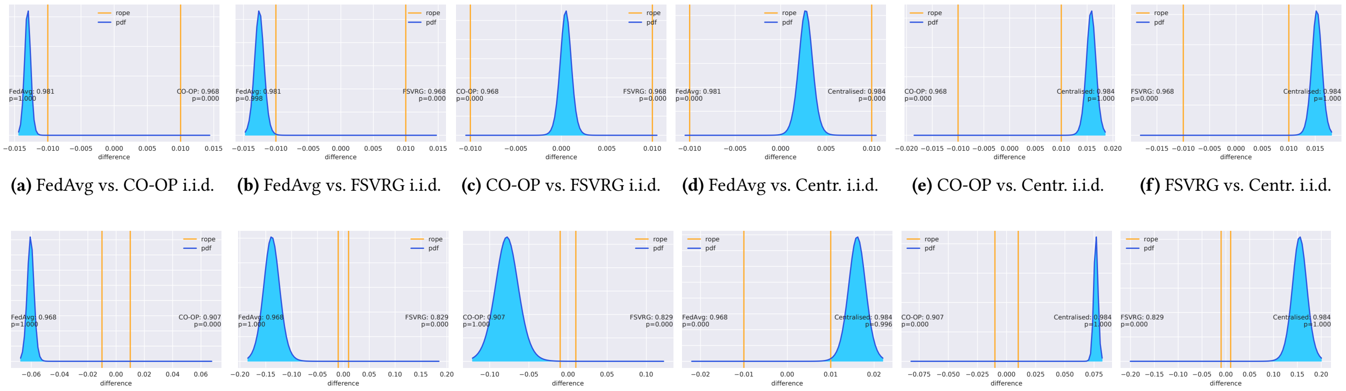
5.1 Hyperparameter searches

With the exception of FSVRG, we performed random searches [2] to find good hyperparameter values. Since FSVRG only has one hyperparameter, a simple grid search was employed instead. The dataset was always MNIST non-i.i.d. This section only provides the values we ultimately used, though additional details with tables and figures are provided in Appendix A.3.

For FedAvg, a total of 56 parameter configurations were evaluated, each for 1,200 communication rounds. First, we searched for η and λ[2] with fixed C, E and B, after which we would try to improve by further exploring C and E. The configuration we ultimately used was: $\eta = 0.088$, $\lambda = 3.2 \cdot 10^{-6}$, $B = 20$, $E = 10$, and $C = 0.1$. The search for FSVRG's stepsize h started at 1 and was increased by 1 until the accuracy no longer improved. The best value we found was $h = 4$. For CO-OP, we tried seven different age filters that were within our restrictions from Sec. 3.3.1; their performance was only marginally different. A subset of η and λ values from the hyperparameter search for FedAvg were reused. The final configuration for CO-OP was: $b_l = 16$, $b_u = 51$, $E = 1$, $B = 20$, $\eta = 0.11$, and $\lambda = 2.2 \cdot 10^{-7}$.

[1] We used the baycomp library to perform Bayesian correlated t-test, which is available at: baycomp.readthedocs.io.

[2] We used Keras' implementation of learning rate decay from [3].



(a) FedAvg vs. CO-OP i.i.d. **(b)** FedAvg vs. FSVRG i.i.d. **(c)** CO-OP vs. FSVRG i.i.d. **(d)** FedAvg vs. Centr. i.i.d. **(e)** CO-OP vs. Centr. i.i.d. **(f)** FSVRG vs. Centr. i.i.d.

(g) FedAvg vs. CO-OP non-i.i.d. **(h)** FedAvg vs. FSVRG non-i.i.d. **(i)** CO-OP vs. FSVRG non-i.i.d. **(j)** FedAvg vs. Centr. non-i.i.d. **(k)** CO-OP vs. Centr. non-i.i.d. **(l)** FSVRG vs. Centr. non-i.i.d.

Figure 3: Comparisons between federated optimization algorithms as well as centralized learning in form of posterior distributions on MNIST i.i.d. and non-i.i.d. Each algorithm ran three iterations of 5-fold cross-validation. Note that the x-axes have different scales.

The centralized approach uses the mini-batch SGD algorithm with all training data available in one place. With centralized learning, a total of 56 random configurations were evaluated in search for good η, λ, and B values. Each configuration trained until it converged. The best configuration we found was: $\eta = 7.03 \cdot 10^{-2}$, $\lambda = 3.77 \cdot 10^{-6}$, and $B = 10$.

5.2 Cross-validation and Bayesian analysis

Results from three iterations of 5-fold cross-validation were used as input to each Bayesian correlated t-test. Several such tests were performed, and the resulting posterior distributions are shown in Fig. 3. The vertical lines show the rope, which we chose to be between $[-0.01, 0.01]$. All probability density functions in Fig. 3 have more than 95 percent of their area within one region. Because this area is interpreted as a probability of relative performance, we are able to summarize our results in Tab. 1.

These comparisons may seem unfair since the federated algorithms are limited by the number of uploads and are therefore further away from converging. While FedAvg showed little improvement after 1,000 global updates, it was unclear how much FSVRG and CO-OP would improve if more communication is allowed. We therefore also experimented with FSVRG and CO-OP using an increased communication budget. In short, these experiments showed that FSVRG eventually rivals FedAvg while CO-OP never managed to reach comparable performance. The Bayesian plots are provided in Appendix A.5.

6 CONCLUSION AND FUTURE WORK

This paper evaluated and compared the federated learning algorithms FedAvg, CO-OP, and FSVRG on the MNIST dataset using Bayesian correlated t-tests. In our experiments with a multi-layer perceptron model, FedAvg showed practically better performance on MNIST when the number of uploads made by clients is limited to 10,000. FedAvg also had comparable performance to centralized learning when using i.i.d. data, but not with non-i.i.d. data.

Our experiments used balanced distributions where all clients were given the same amount of data. However, this does not adequately reflect a heterogeneous federated setting. Investigating performance when client data is unevenly distributed would be a natural extension to our work.

Table 1: Summary of algorithm comparisons, showing if the algorithm in a row is better (+), worse (−), or practically equivalent (=) compared to the algorithm in a column.

		i.i.d.		
	FedAvg	CO-OP	FSVRG	Centr.
FedAvg	×	+	+	=
CO-OP	−	×	=	−
FSVRG	−	=	×	−
		non-i.i.d.		
FedAvg	×	+	+	−
CO-OP	−	×	+	−
FSVRG	−	−	×	−

ACKNOWLEDGMENTS

This work was supported by the project On-board/Off-board Distributed Data Analytics, which is a part of the FFI funding program (DNR 2016-04260) administered by VINNOVA (Sweden’s government agency for innovation). This work was developed in the Fraunhofer Cluster of Excellence »Cognitive Internet Technologies«.

A APPENDIX

A.1 Artificial neural network architecture

We reused the *2NN* architecture found in [13]. It has two hidden layers, where each hidden layer has 200 neurons, each neuron using a ReLu activation function. The output layer uses a softmax function. Moreover, we used the cross-entropy loss function, which is a common choice for neural networks [6, Sec. 6.2]. In total, the network has 199,210 trainable parameters.

A.2 Experimental setup

We used three machines connected with Ethernet in a star topology. All machines ran Ubuntu 16.04 LTS via VirtualBox 5.2. The specifications of each machine are shown in Tab. 2.

Table 2: Specifications of our virtual machines.

Machine	CPU	Memory (GiB)
A	i7-6700K	23.5
B	i7-6700K	23.5
C	i7-7700K	23.4

Clients and the server are mainly implemented as separate Erlang nodes. Erlang handles all client-server communication, whereas machine learning computations are relayed to Python processes via JSON files.

A.3 Hyperparameter searches

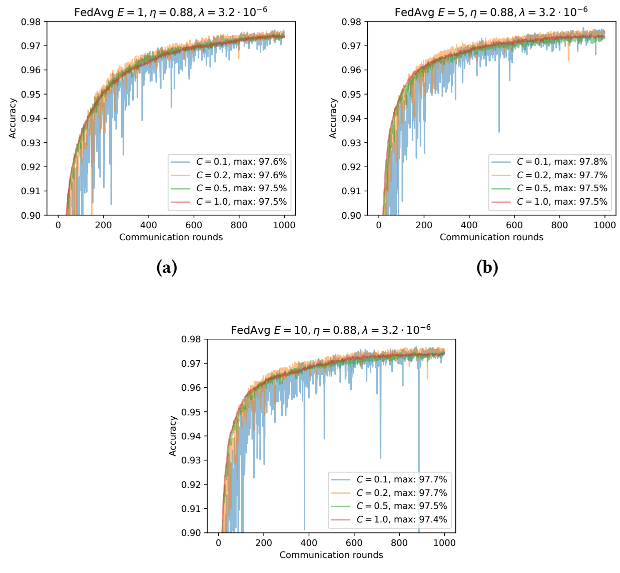
This section shows some of the hyperparameter searches we conducted. Tab. 3 and Fig. 4 are from FedAvg, Tab. 4 from FSVRG, and Fig. 5 from CO-OP. Only a selected portion of all the runs are shown in Tab. 3.

Table 3: Random search for good values of η and λ for FedAvg. The best values are boldfaced. The other parameters were $C = 0.1$, $E = 5$, $B = 20$, and non-i.i.d. MNIST.

η	λ	max(acc)	min(err)	avg(acc)
0.05	1.6e-06	0.9767	0.101	0.9582
0.068	2.8e-07	0.978	0.109	0.96
0.088	3.2e-06	0.9788	0.09794	0.9638
0.098	2.4e-08	0.979	0.1325	0.9562
0.11	2.2e-07	**0.9792**	**0.09315**	**0.9658**
0.14	7.8e-04	0.6962	0.8643	0.09896

Table 4: A linear search for FSVRG's stepsize parameter h. The best values are boldfaced. The dataset was non-i.i.d. MNIST.

h	max(acc)	min(err)	avg(acc)
1.0	0.975	0.07994	**0.9314**
2.0	0.9799	0.07058	0.9159
3.0	0.9801	0.06742	0.9157
4.0	**0.9816**	**0.0663**	0.9135
5.0	0.9811	0.06648	0.9081



(a) **(b)**

(c)

Figure 4: Single runs of FedAvg with different C and E values; $B = 20$. The dataset was MNIST non-i.i.d.

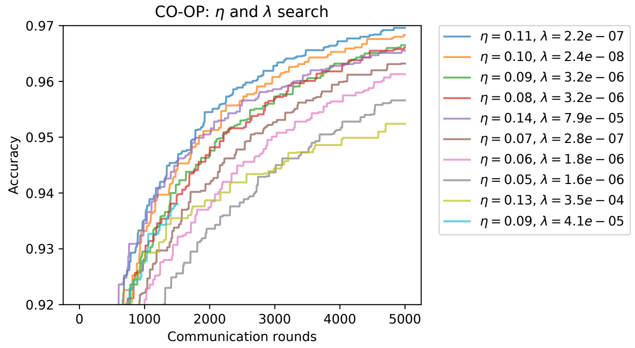


Figure 5: Monotonic representations of training with CO-OP using MNIST non-i.i.d. for different learning rates and decays. The legend is sorted from highest to the lowest final accuracy. The other parameters were $b_l = 16$, $b_u = 51$, $E = 1$, and $B = 20$.

A.4 Proof that CO-OP may deadlock

Theorem A.1. *It is possible for the CO-OP algorithm to deadlock if $b_u < 2b_l$ and the number of clients $K \geq b_u - b_l + 1$.*

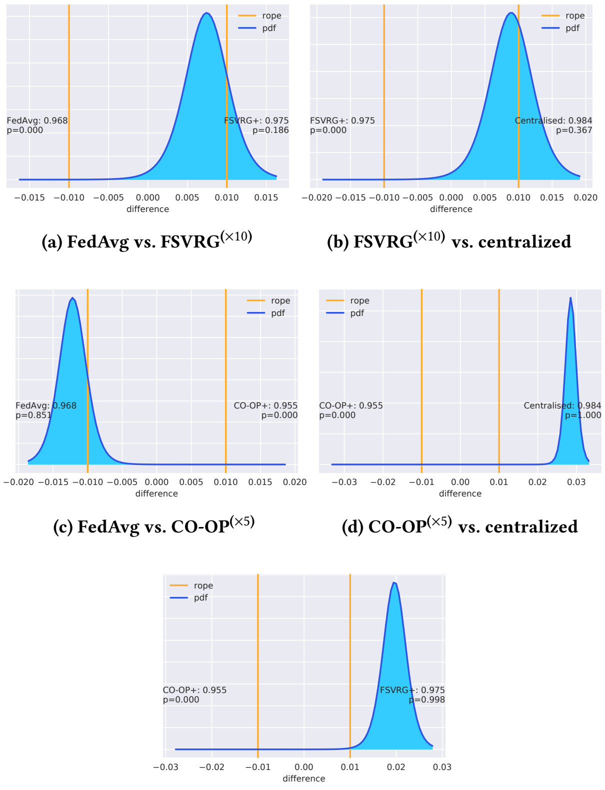
Proof. Initially, $a = b_l$ and $\forall k : a_k = 0$. To reach a state where $a = b_u + 1$, we need at least $b_u - b_l + 1$ updates. Assume that these updates are performed by $b_u - b_l + 1$ distinct clients. Then, $a = b_u + 1$ and the $b_u - b_l + 1$ clients who updated have ages $b_l + 1, b_l + 2, \ldots, b_u + 1$. Let f be the client who performed the very first update, hence $a_f = b_l + 1$. From the algorithm, it follows that client f is overactive if the following holds:

$$\begin{aligned} a - a_f &< b_l \Leftrightarrow \\ (b_u+1)-(b_l+1) &< b_l \Leftrightarrow \\ b_u - b_l &< b_l \Leftrightarrow \\ b_u &< 2b_l \quad . \end{aligned} \tag{3}$$

Assuming (3) holds, f, and all of the other $b_u - b_l$ clients who have updated so far, are regarded as overactive. Recall that there are $K - (b_u - b_l + 1)$ clients that still have $a_k = 0$. If these clients read the most recent $a = b_u + 1$, they are considered outdated and set their $a_k = a$. However, we are now in a state where all clients are considered overactive. Thus, it is possible for CO-OP to deadlock if $b_u < 2b_l$. □

A.5 Extended number of uploads

When FSVRG is given ten times more uploads, it converges to a performance that is practically equivalent or better compared to FedAvg, see Fig. 6. CO-OP is still inferior in terms of accuracy if given five times more uploads.



(a) FedAvg vs. FSVRG$^{(\times 10)}$

(b) FSVRG$^{(\times 10)}$ vs. centralized

(c) FedAvg vs. CO-OP$^{(\times 5)}$

(d) CO-OP$^{(\times 5)}$ vs. centralized

(e) CO-OP$^{(\times 5)}$ vs. FSVRG$^{(\times 10)}$

Figure 6: Comparisons between the federated optimization algorithms on non-i.i.d. MNIST in form of posterior. Each algorithm ran two iterations of 5-fold cross-validation. Note that the x-axes have different scales.

REFERENCES

[1] Alessio Benavoli, Giorgio Corani, Janez Demšar, and Marco Zaffalon. 2017. Time for a Change: a Tutorial for Comparing Multiple Classifiers Through Bayesian Analysis. *Journal of Machine Learning Research* 18, 77 (2017), 1–36. http://jmlr.org/papers/v18/16-305.html

[2] James Bergstra and Yoshua Bengio. 2012. Random Search for Hyper-parameter Optimization. *Journal of Machine Learning Research* 13, 1 (Feb. 2012), 281–305. http://dl.acm.org/citation.cfm?id=2503308.2188395

[3] François Chollet et al. 2018. Keras' learning rate decay implementation. Retrieved August 17, 2018 from https://github.com/keras-team/keras/blob/29a22a8d59b5e2c4282f1e7f664d82595049eb9d/keras/optimizers.py#L178

[4] Giorgio Corani and Alessio Benavoli. 2015. A Bayesian approach for comparing cross-validated algorithms on multiple data sets. *Machine Learning* 100, 2 (01 Sept. 2015), 285–304. https://doi.org/10.1007/s10994-015-5486-z

[5] European Commission. 2018. What data can we process and under which conditions? Retrieved April 26, 2018 from https://ec.europa.eu/info/law/law-topic/data-protection/reform/rules-business-and-organisations/principles-gdpr/what-data-can-we-process-and-under-which-conditions_en

[6] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

[7] Hitachi Data Systems. 2015. The Internet on Wheels and Hitachi, Ltd. White Paper.

[8] Jonghwan Hyun, Youngjoon Won, Eunji Kim, Jae-Hyoung Yoo, and James Won-Ki Hong. 2016. Is LTE-Advanced really advanced?. In *IEEE/IFIP Network Operations and Management Symposium (NOMS)*. 703–707. https://doi.org/10.1109/NOMS.2016.7502881

[9] Mathias Johanson, Stanislav Belenki, Jonas Jalminger, Magnus Fant, and Mats Gjertz. 2014. Big Automotive Data: Leveraging large volumes of data for knowledge-driven product development. In *IEEE International Conference on Big Data (Big Data)*. 736–741. https://doi.org/10.1109/BigData.2014.7004298

[10] Jakub Konecný, H. Brendan McMahan, Daniel Ramage, and Peter Richtárik. 2016. Federated Optimization: Distributed Machine Learning for On-Device Intelligence. arXiv:1610.02527

[11] Yann LeCun and Corinna Cortes. 1998. The MNIST database of handwritten digits. http://yann.lecun.com/exdb/mnist/

[12] David Martín-Sacristán, Jose F. Monserrat, Jorge Cabrejas-Peñuelas, Daniel Calabuig, Salvador Garrigas, and Narcís Cardona. 2009. On the Way towards Fourth-Generation Mobile: 3GPP LTE and LTE-Advanced. *EURASIP J. Wireless Commun. and Networking* 2009, 1 (03 Aug 2009). https://doi.org/10.1155/2009/354089

[13] H Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, et al. 2016. Communication-efficient learning of deep networks from decentralized data. arXiv:1602.05629

[14] Yushi Wang. 2017. *CO-OP: Cooperative Machine Learning from Mobile Devices*. Master's thesis. Dept. Elect. and Comput. Eng., Univ. Alberta, Edmonton, Canada.