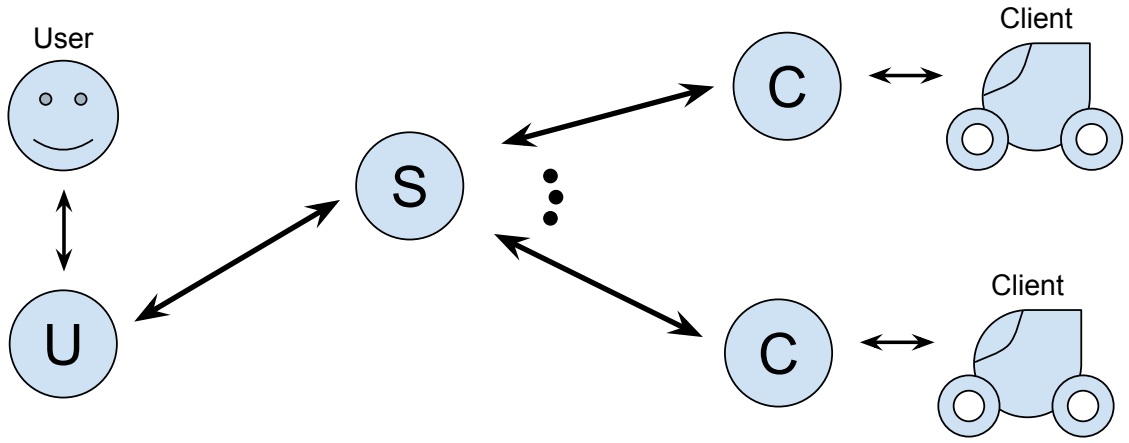
CHALMERS
UNIVERSITY OF TECHNOLOGY




# Evaluating the Performance of Federated Learning

## A Case Study of Distributed Machine Learning with Erlang

Master's thesis in Computer Science – Algorithms, Languages and Logic

ADRIAN NILSSON
SIMON SMITH

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2018

Master's thesis 2018

# Evaluating the Performance of Federated Learning

A Case Study of Distributed Machine Learning with Erlang

ADRIAN NILSSON
SIMON SMITH


CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
Gothenburg, Sweden 2018

Evaluating the Performance of Federated Learning
A Case Study of Distributed Machine Learning with Erlang
ADRIAN NILSSON
SIMON SMITH



Supervisor: Nicholas Smallbone, Computer Science and Engineering
Advisor 1: Gregor Ulm, Fraunhofer-Chalmers Research Centre for Industrial Mathematics
Advisor 2: Emil Gustavsson, Fraunhofer-Chalmers Research Centre for Industrial Mathematics
Examiner: Peter Damaschke, Computer Science and Engineering

Master's Thesis 2018
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: A graph representation of the framework we built to perform Federated Learning, using Erlang nodes for communicate between U, S and, C. The edge devices use Python for computations and user interactions.

Typeset in LaTeX
Gothenburg, Sweden 2018

Evaluating the Performance of Federated Learning
A Case Study of Distributed Machine Learning with Erlang
ADRIAN NILSSON
SIMON SMITH
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

# Abstract

An alternative environment for distributed machine learning has recently been proposed in what is called Federated Learning. In Federated Learning, a global model is learnt by aggregating models that have been optimised locally on the same distributed clients that generate training data. Contrary to centralised optimisation, clients in the setting of Federated Learning can be very large in number and are characterised by challenges of data and network heterogeneity. Examples of clients include smartphones and connected vehicles, which highlights the practical relevance of this approach to distributed machine learning.

We compare three algorithms for Federated Learning and benchmark their performance against a centralised approach where data resides on the server. The algorithms covered are Federated Averaging (FedAvg), Federated Stochastic Variance Reduced Gradient (FSVRG), and CO-OP. They are evaluated on the MNIST dataset using both IID and non-IID partitionings of the data. Our results show that, among the three federated algorithms, FedAvg trains the model with the highest accuracy regardless of how data was partitioned. Our comparison between FedAvg and centralised learning shows that they are practically equivalent when IID data is used, but the centralised approach outperforms FedAvg with non-IID data. We recommend FedAvg over FSVRG and see practical benefits for an asynchronous algorithm, such as CO-OP.

# Acknowledgements

We are grateful for all the great discussions, the fruits, the fredagsfika, and the tea at FCC. Also, a big thank you to all the feedback we have had on the report from Andreas Lindhé, Robert Gustafsson, Gregor Ulm, and Nick Smallbone.

Adrian Nilsson, Gothenburg, June 2018

Simon Smith, Gothenburg, June 2018

# Contents

# 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 behaviour and driving environments. Current approaches to data analysis on a vehicle fleet involve sending compressed sensor-data from each vehicle to a central server that carries out data analysis tasks [1]. However, a modern car can produce hundreds of gigabytes of data each day [2], which means that data transfer and central data storage are infeasible in practice. Furthermore, user data is private by nature, which raises privacy concerns about transferring and storing such data in a central server.

With a recent approach called *Federated Learning* [3], 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 each client, using their own data. Consequently, no user data are stored in a central server.

In Federated Learning, 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. This raises the question if Federated Learning can attain performance comparable to a centralised approach. Addressing this question is the main purpose of our work. Moreover, our performance evaluation includes several Federated Learning algorithms to determine if any algorithm should be preferred under different circumstances.

## 1.1 Motivating Federated Learning

The Federated Learning problem is different from other distributed problems (e.g. data centre distribution) in several aspects. In theory, both approaches aim to optimise their learning objective. In practice, however, Federated Learning algorithms have to account for the fact that communication with edge devices takes place over unreliable networks with very limited upload speeds. Federated Learning therefore aims to minimise the communication cost and still produce good models. In the

distributed setting of a data centre, where one has access to fast LAN connections, communication cost is less of an issue.

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 optimisation algorithms typically assume that [3, 4]:

- Data is evenly distributed across clients.
- Client-side data are Independently and Identically Distributed (IID) samples from the overall distribution.
- The number of clients is much smaller than the average number of locally available training examples per client.

In contrast, algorithms for Federated Learning *cannot* make these assumptions. To see why, consider our motivating use case where clients are vehicles in a fleet. Since vehicles may be used to varying degrees and therefore access different amounts of local data, one cannot assume that data is evenly distributed. Also, individual vehicles clearly do not contain identically distributed data — one only has to consider that any vehicle's behaviour is dependent on its driver to realise this. It is, however, more difficult to reason about the ratio between the fleet size and the number of training examples available on each vehicle. Although, if we assume a fleet size in the order of thousands of vehicles, and that local training examples are inferred from user interaction (e.g. braking), then it is a plausible scenario that there are many more vehicles in the fleet than the average number of inferred training examples.

## 1.2 Goals

The overall goal of our thesis is to evaluate how well Federated Learning performs on a particular classification problem. We further break down our overall goal into three parts:

1. Assessing the practicality of Federated Learning, in terms of performance, by comparing it with centralised learning.
2. Drawing conclusions about what federated optimisation algorithm should be preferred under different circumstances by comparing different algorithms.
3. Building a framework for Federated Learning.

Note that Federated Learning only refers to the specific distributed setting — the actual optimisation can be carried out in numerous ways. A proper performance evaluation of Federated Learning should therefore consider multiple optimisation algorithms. FedAvg, FSVRG, and CO-OP are three of the few general federated

optimisation algorithms in existence. These three algorithms are implemented and compared in this thesis.

## 1.3 Approach

Our evaluation focuses on non-convex optimisation, specifically classification tasks using ANNs, in the setting of Federated Learning. We refer to this simply as *federated optimisation*, although this is somewhat imprecise since Federated Learning is not inherently constrained to optimising ANN models.

We benchmark on the MNIST dataset of handwritten digits [5]. Since communication in Federated Learning is much more expensive than computation, it is desirable to have as little communication as possible. Therefore, we define performance in Federated Learning 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.

## 1.4 Affiliation

This thesis was carried out at the System and Data Analysis department of Fraunhofer-Chalmers Research Centre for Industrial Mathematics (FCC) as a part of the ongoing research project On-board/Off-board Distributed Data Analytics (OODIDA). The OODIDA project is funded in part by VINNOVA's funding program Fordonsstrategisk Forskning och Innovation (FFI) (DNR 2016-04260). The communication framework is based on a prototype from the OODIDA project.

# 2

# Background

This chapter covers the background needed to fully motivate Federated Learning as well as practices for algorithm evaluation and comparison. We initially review the optimisation problem at the heart of much of machine learning and the problems associated with solving that problem in a big data setting. After formulating the distributed version of the optimisation problem, we continue by motivating and describing Federated Learning. We conclude with a review of commonly used performance measures and practices for algorithm evaluation and comparison.

## 2.1 The optimisation problem

The algorithms we consider are designed to optimise a finite-sum objective

$$\min_{w \in \mathbb{R}^d} f(w) \qquad f(w) := \frac{1}{n} \sum_{i=1}^{n} f_i(w). \tag{2.1}$$

Here, $w$ 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 labelled examples, often referred to as *training examples*. The objective function $f(w)$ is defined by the high-dimensional vector $w$ conditioned on our $n$ labelled examples. The problem can now be interpreted as finding the $w$ which minimises the average loss over all $n$ training examples. We refer to the process of finding an optimal $w$ as *training*.

A way to model the optimisation problem stated in (2.1) is to let $w$ be the weights of an ANN. Using such an ANN model, a successful approach to solving the problem is to apply Stochastic Gradient Decent (SGD) [6, 7] and compute loss gradients $\nabla f_i(w)$ using backpropagation [8]. The only missing piece is the objective function, which must be chosen appropriately by the practitioner.

## 2.2 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 each day. For example, a moderately sized fleet of 1000 test vehicles is estimated to be able to produce data in the order of terabytes each day [1], and Hitachi [2] 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*. Big data analytics hold promise for more accurate validation and testing models, as well as improving consumer experience.

The SGD algorithm requires that all training examples are available on the machine that trains the ANN. This centralised approach has limitations in a big data setting, where data is generated in huge volume and with great velocity. Notably, if data is generated on edge devices, then wireless data transfer over cellular networks becomes the principal bottleneck. For instance, transferring 25GB/hour translates to a constant upload throughput of $\approx 7\text{MB/s} = 56\text{Mb/s}$. While the LTE and LTE-Advanced standard of 4G specifies peak upload data rates of 50Mb/s and 500Mb/s respectively [9], a recent case study on data rates in central South Korea [10] shows that practical performance is far from ideal. 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. This suggests that transferring all generated data is infeasible in practice, and even if it were, we would quickly exhaust the consumers' data plans in the case of phones. Also, besides the challenge of data transfer, simply storing such vast amounts of data centrally will quickly become a problem in itself.

Besides the technical challenges posed by big data, collecting and storing large amounts of data centrally is problematic from a privacy perspective. The General Data Protection Regulation (GDPR) recently came into effect in the EU, which has several implications for how personal data can be collected and processed. Data regarding, for instance, driving patterns is indeed personal if it can be connected to an individual in any way. GDPR has principles for purpose and storage limitation as well as data minimisation [11]. Personal data can only be processed with explicit consent for specific, known purposes (purpose limitation), and only the minimal amount of data required to fulfil those purposes may be collected (data minimisation). Moreover, data should be removed once its purpose has been fulfilled (storage limitation). A White House report from the Obama administration [12] gives as similar proposal in what they term *focused collection*. Performing big data analytics on consumer data while respecting data privacy laws is therefore not a straightforward process.

## 2.3 Distributed optimisation

When the number of training examples becomes too large to store on one computer, which is the case in a big data context, we have to distribute the computation to multiple computers. Distributing the data and computational burden to multiple computers leads us to reformulate the objective function $f(w)$ from (2.1) to (2.2). Assume there are $K$ clients to 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 $k$. 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 $F_k(w)$:

$$f(w) = \sum_{k=1}^{K} \frac{n_k}{n} F_k(w), \quad \text{where} \quad F_k(w) := \frac{1}{n_k} \sum_{i \in \mathcal{P}_k} f_i(w). \tag{2.2}$$

However, existing approaches focus mainly on the case of data centre optimisation where computation rather than communication is the primary bottleneck. Distributed data centre optimisation typically requires control over the data distribution since these approaches rely on balanced, i.e. equal $n_k$ for all $k$, and IID data assumptions. These assumptions are too strong when we want to perform learning tasks on a heterogeneous ecosystem of edge devices.

## 2.4 Federated Learning

Federated Learning proposes an alternative approach where a set of clients performs learning tasks locally and only communicate an updated model to a coordinating server. More specifically, Federated Learning proposes to have a server learn a shared global model by aggregating locally trained models from a possibly very large number of clients whose data transfer capabilities are limited. A contrast is drawn to the distributed setting in that the *federated setting* features a massive number of clients, unbalanced and non-IID client data, and high communication costs. Note that the learning algorithm is not made explicit by the federated setting. Although our focus is on optimising an ANN model, Federated Learning can be applied to other models as well, for instance a Support Vector Machine (SVM) or even a simple regression model.

A synchronous Federated Learning algorithm typically performs one communication round in four steps [13]. First, some, possibly all, clients are selected to participate, all of which download the current global model. Second, participating clients use their local data to compute a local model update. Third, participating clients upload their local model update to the server. Finally, the server aggregates the model

updates received from all participating clients to produce a new global model. Once the final step is completed, the received updates are discarded.

The federated approach has advantages over the centralised alternative when it comes to data privacy. For instance, Federated Learning applies GDPR's data minimisation principle [11] since only the learnt 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 [11] .

## 2.5 Algorithm performance measures

Central to any evaluation is the question of how performance is defined. We have already established that Federated Learning algorithms aim to optimise for communication efficiency in the sense that we want to achieve high performance in few communication rounds. Defining performance is, however, not self-evident as there is an abundance of suggested classifier performance measures within the literature [14, 15, 16]. In the following subsections, we describe some commonly used measures to understand how this choice affects how classifiers are evaluated.

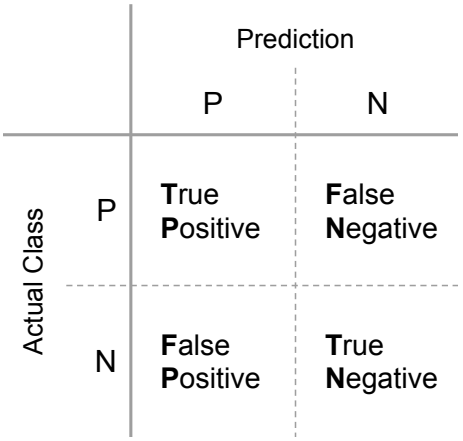
### 2.5.1 Classification accuracy

Classification accuracy, or simply accuracy, refers to the percentage of correctly classified examples from the test set. Accuracy is arguably the historically dominating performance measure within machine learning research. For instance, between the years 1999 to 2003, about two thirds of the papers accepted at the International Conference on Machine Learning that compared classifiers over multiple datasets used accuracy as their only scoring metric [17, Tab. 1].

While accuracy has a clear and intuitive interpretation, its use as an evaluation metric for comparing machine learning algorithms has been criticised as being inadequate for real-world tasks [18, 19]. Accuracy is unsuitable in domains where class skewness is prevalent and where it is more important to correctly identify certain classes. An example of such a domain is medicinal screening, for instance cancer detection, which we will use as a running example in the remainder of this chapter. To be clear, a positive prediction for cancer detection means that the patient is predicted to have cancer. If we just measure accuracy on this task, then, due to severe class skew, a naive classifier that always gives negative predictions is likely to perform better than a classifier that actually manages to identify cancer in some patients.

### 2.5.2 Precision and recall

It is easier to think about precision and recall in the case of binary classification, where one class is considered to be positive and the other negative. Precision and recall are then concisely defined by a *confusion matrix*. Fig. 2.1 shows a confusion matrix for binary classification as well as some common metrics derived from it. From Fig. 2.1, we see that precision is defined as the number of true positives (the number of correct classifications to the positive class) divided by the total number of positive predictions. The recall measure is sometimes referred to as the *true positive rate*, which is the number of true positives divided by all positive examples in the test set.




**(a)**

| Measure | Formula |
|---|---|
| Accuracy: | $\frac{TP+TN}{TP+TN+FP+FN}$ |
| Precision: | $\frac{TP}{TP+FP}$ |
| Recall: | $\frac{TP}{TP+FN}$ |
| FP-rate: | $\frac{FP}{FP+TN}$ |

**(b)**

**Figure 2.1:** A so-called confusion matrix for binary classification is shown in 2.1a, from which a number of common performance measures are derived in 2.1b.

In the case of cancer prediction, precision represents how often we correctly identify cancer in all patients. Recall instead measures how often we correctly identify cancer in all patients that actually have cancer. However, note that a naive classifier that always gives positive predictions will have perfect recall. The recall measure is therefore misleading to use on this naive classifier. On the other hand, if only precision is used, then we have no information about how often we incorrectly say that healthy patients have cancer, which is highly relevant. Therefore, precision and recall are often used jointly. Also, it is common to combine precision and recall to a single value called the F-score [15].

### 2.5.3 Receiver operating characteristic

To measure accuracy, precision, or recall, a classifier only needs to output a class label. However, classifiers often have probabilistic outputs, for instance the output of a softmax function in an ANN classifier, that give an estimated probability of examples belonging to either class. This information is ignored in these measures

since we simply assign the class label with the highest probability estimate. Other measures, such as Receiver Operating Characteristic curve (ROC), instead exploit the extra information given by probability estimates.

An ROC curve visualises the relationship between true positive rate and false positive rate over all possible acceptance thresholds. The ROC curve has some attractive properties: it visualises classifier performance while being insensitive to class distribution (and hence class skewness) as well as costs of misclassification [20]. However, comparing ROC curves often requires an analysis. Unless a classifier dominates, i.e. lies above all other curves, in ROC space, it may not be clear what classifier, if any, has better performance. Also, a fundamental limitation with ROC analysis is that it only applies to binary classifiers.

### 2.5.4 Area under the ROC curve

A way to reduce the ROC curve to a single value is to calculate the Area Under the Receiver Operating Characteristic curve (AUROC), often referred to as just AUC. This measure has a known statistical interpretation: "[AUC] is equivalent to the probability that a randomly chosen member of one class has a smaller estimated probability of belonging to the other class than has a randomly chosen member of the other class" [21, p.184].

It has been formally argued that AUC should replace accuracy when comparing classifiers and their performance [22]. However, more recent findings by [23] show that AUC is, for practical purposes, an incoherent measure of performance when comparing classifiers, and they therefore discourage its use. A coherent alternative to AUC is given in [24], but its use is not yet widespread.

### 2.5.5 Multi-class considerations

Because ROC and AUC only deal with binary classifiers, they are not directly applicable when we consider multi-class classifiers. Nevertheless, some alternative measures have been proposed for multi-class classifiers, for instance the M-measure [25]. The M-measure is an alternative to AUC for multi-class classification that maintains AUC's properties of being insensitive to class skewness and error costs [21]. However, since the M-measure is completely based on AUC, it also inherits the AUC measure's issue of incoherency.

## 2.6 Algorithm evaluation

When a new algorithm is proposed, it is common to show that it improves on previous algorithms in some aspect. The natural question to ask is if algorithm A is better than algorithm B, or how probable it is that algorithm A produces a better classifier compared to algorithm B. For classification tasks, a better algorithm is commonly interpreted as producing more accurate classifiers, though other performance measures may be used.

Ultimately, we are concerned with how well classifiers generalise to new data; how accurately our model can classify unseen examples. Because we cannot know what future data a classifier will be exposed to in practice, we must estimate the generalisation performance using the available data. Therefore, it is important to maintain a test set of labelled examples that is never exposed to the training algorithm. However, a small test set will introduce increased uncertainty in the estimate of generalisation performance [26]. Unless the dataset has hundreds of thousands of examples, procedures such as cross-validation can be applied to make better use of the available data at the cost of additional computation [26].

In what is known as $k$-fold cross-validation [26, Sec. 5.3.1], the available data is partitioned into $k$ equally sized, or near-equally sized, subsets, so called *folds*. The training procedure is then carried out in $k$ separate runs. In each run, a fold is held out and acts as the test fold, leaving the other $k - 1$ folds to be used for training. After all $k$ runs have finished, each fold will have acted as the test fold in exactly one run. The average performance across all $k$ runs is computed and given as the final estimate of the generalisation performance.

Once an estimate of generalisation performance has been acquired for each of the classifiers in question, we can look at their difference to determine which one has the better performance. As a final step, it is not uncommon to statistically verify that a given classifier indeed improves performance, that is, the results were not random.

### 2.6.1 Null hypothesis significance testing

The desire to statistically verify that the observed differences in classifier performance are significant led to the adoption of *null hypothesis significance tests*. The null hypothesis is that the compared algorithms are equivalent. Conversely, the alternate hypothesis is that the algorithms are not equivalent. A correlated t-test can be used to evaluate the performance of two classifiers on a single dataset. The t-test produces a $p$-value that represents the probability of obtaining performance differences equal to, or greater than, the observed differences assuming the null hypothesis is true. If $p \leq 0.05$, then it is common practice to state that the algorithms gave significantly different classifiers on the tested dataset.

The objective of significance testing in supervised machine learning is to quantify the probability that two classifiers give different performance. However, the t-test employed in frequentist null hypothesis significance tests tends to be misused to draw incorrect conclusions [27]. Specifically, $(1 - p)$ is sometimes incorrectly interpreted as the probability of the alternate hypothesis, where in fact the $p$-value gives no information about how probable the null and alternate hypotheses are. The $p$-value actually represents the probability of the observed results assuming the null hypothesis is true, which is not the probability of the hypothesis based on the observed results that we want to know. This and several other issues with frequentist null hypothesis significance testing lead the authors of [27] to discourage its continued use for evaluating the performance of classifiers.

### 2.6.2 Bayesian approaches

The practice of null hypothesis significance testing is discouraged by [27] and they recommend to use Bayesian approaches instead. For example, if we compare the two classifiers that two different algorithms produced on the same dataset, then a Bayesian correlated t-test [28] can be employed instead of the usual correlated t-test. The output of such a test is a posterior probability distribution that represents the mean difference in performance, e.g. accuracy, between the tested classifiers. This posterior can be used to infer a probability that one classifier is better than the other, which is what we wanted to, but could not, infer from a frequentist correlated t-test.

While two classifiers can have very similar performance, they never show exactly equivalent performance. Therefore, it is known beforehand that the frequentist null hypothesis of equivalence is false. The Bayesian approach offers a way to reason about near, or practical, equivalence that null hypothesis significance testing cannot. For instance, if the difference in accuracies between two classifiers is less than 1%, then it might be sensible to view them as practically equivalent. We then define a *region of practical equivalence* (rope) between $[-0.01, 0.01]$. The area under the posterior function in this region then represents the probability that the mean difference in accuracy is within $\pm 1\%$, which is more informative compared to rejecting a null hypothesis that is known to be false.

# 3

# Overview of Algorithms

Our evaluation of Federated Learning algorithms includes the synchronous algorithms Federated Averaging (FedAvg) and Federated Stochastic Variance Reduced Gradient (FSVRG) as well as the asynchronous CO-OP algorithm. This chapter presents the full algorithms as well as a more high-level description of each algorithm. We further highlight important considerations for implementing these algorithms. Because this chapter makes heavy use of notation, a table of notation is included in Appendix A.1 for convenience.

## 3.1 Federated Averaging

Federated Averaging (FedAvg) is based on maintaining a shared global model which is periodically updated by averaging models that have been trained locally on clients. Training is orchestrated by a central server which hosts the shared global model $w_t$. However, the actual optimisation is done locally on clients using, for instance, 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 $\eta$, and possibly a learning rate decay[1] $\lambda$ [3]. The parameters $B, \eta$, and $\lambda$ are commonly used when training with SGD. $E$ is also commonly used with SGD, but here $E$ stands for the total number of iterations through the same data *before the global model is updated*.

The algorithm starts by randomly initialising the global model $w_t$. 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 $w_{t+1}^k$ to the server, which then generates the new global model $w_{t+1}$ by computing a weighted sum of all received local models. The complete algorithm is given in Alg. 1.

[1] We used Keras' implementation of learning rate decay from: github.com/keras-team/keras.

**Algorithm 1:** FederatedAveraging

1 initialise $w_0$
2 **for** *each round* $t = 0, 1, \ldots$ **do**
3 $\quad m \leftarrow \max(\lfloor C \cdot K \rfloor, 1)$
4 $\quad S_t =$ random set of $m$ clients
5 $\quad$ **for** *each client* $k \in S_t$ ***in parallel*** **do**
6 $\quad\quad w_{t+1}^k = \texttt{ClientUpdate}(k, w_t)$
7 $\quad w_{t+1} = \sum_{k=1}^{K} \frac{n_k}{n} w_{t+1}^k$ ; `// Update global model`

Note that the sum of model updates in the original FedAvg algorithm (Alg. 1, line 7) goes over all $K$ clients, even though only a subset $S_t$ of all clients has computed an updated model. This raises the question of what $w_{t+1}^k$ is for the clients that were not chosen to participate in a particular iteration. We have identified three possible interpretations of how $w_{t+1}^k$ is defined for non-participating clients, which leads to three different ways of updating $w_{t+1}$. These are presented in (3.1), (3.2), and (3.3) in a form that highlights how they differ from each other. They are therefore not necessarily given in their most compact form.

$$w_{t+1} = \sum_{k \in S_t} \frac{n_k}{n} w_{t+1}^k + w_t \sum_{k \notin S_t} \frac{n_k}{n} \tag{3.1}$$

The update in (3.1) uses the current global model as a substitute for client updates, that is, $w_{t+1}^k = w_t$ for $k \notin S_t$. This approach might be viable when $w_0$ is initialised to a pre-trained model or when using larger values of $C$. Otherwise, if $w_0$ is initialised randomly and $C < 0.5$, then this update would initially give more weight to the random global model than the client updates.

$$w_{t+1} = \sum_{k \in S_t} \frac{n_k}{\mu} w_{t+1}^k \tag{3.2}$$

The perhaps most straightforward update is given in (3.2). Here, only the updated local models from the selected clients are considered. In other words, $w_{t+1}^k = \mathbf{0}$ for $k \notin S_t$. For this update to make sense, $\mu$ should be the amount of data on all selected clients, i.e. $\mu = \sum_{k \in S_t} n_k$. Otherwise the weights will not sum to unity. A simplified version of (3.2) that further assumes evenly distributed data ($\forall k, n_k = G$, where $G$ is some constant) is used in the implementation of FedAvg that CO-OP compares itself to [29, p. 14].

$$w_{t+1} = \sum_{k \in S_t} \frac{n_k}{n} w_{t+1}^k + \sum_{k \notin S_t} \frac{n_k}{n} w_{t+1}^k, \quad w_{t+1}^k = w_t^k \text{ for } k \notin S_t \tag{3.3}$$

The third interpretation, given in (3.3), is somewhat more subtle. The intuition is that the second term considers the most recent update from all clients who have ever participated in any previous communication round. An underlying assumption is that $w_t^k = \mathbf{0}$ until the first time client $k$ contributes an update. Again, this only makes sense if all weights sum to unity, which in this case implies that $n = \sum_{k=1}^{K} n_k$ should be used as the denominator.

## 3.2 Federated Stochastic Variance Reduced Gradient

Federated Stochastic Variance Reduced Gradient (FSVRG) [4] accounts for the challenges of Federated Learning by adopting a distributed approach to the centralised SVRG algorithm. 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 $n_k$, $h_k = h/n_k$. 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 [4, 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 initialise their local model $w_t^k$ and local step-size $h_k$. After creating a random permutation of their local data, clients will iteratively perform $n_k$ SVRG updates, leveraging the full gradient previously computed, with a client specific stepsize $h_k$. Finally, when all clients have computed and uploaded their final $w_{t+1}^k$, the server merges all $w_{t+1}^k$ to form a new global model $w_{t+1}$, similar to FedAvg.

**Algorithm 2:** Federated SVRG

initialise $w_0$
$h \leftarrow$ 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**
    initialise: $w_{t+1}^k \leftarrow w_t$, 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**
      $w_{t+1}^k \leftarrow w_{t+1}^k - h_k \left( \nabla f_{i_s}(w_{t+1}^k) - \nabla f_{i_s}(w_t) + \nabla f(w_t) \right)$
  $w_{t+1} = \sum_{k=1}^{K} \frac{n_k}{n} w_{t+1}^k$ ; `// Update global model`

FSVRG in its original form [4, Alg. 4] is primarily concerned with sparse data in the sense that some features are seldom represented in the data set, 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 $w$ is the same as the dimension of the input $x$ in the Support Vector Machine (SVM) model they consider. But the number of weights in a neural network model is generally much larger than the input dimension. Since we cannot apply these scaling matrices, they are excluded from Algorithm 2.

## 3.3 CO-OP

Whereas FedAvg and FSVRG rely on synchronised model updates, CO-OP [29] proposes an asynchronous approach. Contrary to FedAvg, CO-OP eagerly merges any received client model with the global model. Rather than directly averaging models, the merging between a local and the global model is performed via a weighting scheme based on a measure of the models age difference. This is motivated by the fact that in an asynchronous framework, some clients will train on outdated models whereas others will train on more up-to-date models.

Wang [29] uses $n$ and $n_k$ to denote the age of the global model and the age of client $k$'s local model respectively. To avoid notational confusion with the number of training examples, we use $a$ and $a_k$ instead. The age difference is then simply $a - a_k$. 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*.

The training procedure is as follows. Each client has its own training data, and performs $E$ rounds of an optimisation 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, then the client reconciles with the global model and starts over. Should the local model be overactive, then the client just continues training. Otherwise, the local model is uploaded to the server for merging.

The pseudocode of CO-OP is presented in Algorithm 3. Note that CO-OP inherits all hyperparameters from its underlying optimisation algorithm in line 5. Since we use SGD, CO-OP also has a learning rate, a learning rate decay, as well as the parameter $B$.

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 [29] aggregated data into a batch at certain randomised time intervals. In Wang's method, a client could then only access one batch at a time. For us, this means that our implementation trained on more data and in the same way as our implementations for the other algorithms.

**Algorithm 3:** CO-OP

`// Initialisation:`

$w = w_1 = ... = w_K \leftarrow w_0$

$a \leftarrow b_l$

$a_1 = ... = a_K \leftarrow 0$

`/* Each client` $k$ `performs the following independently: */`

**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**

    `// Client is overactive`

    **continue**

  **else**

    `// Normal update`

    Upload $w_k, a_k$ to the server. The server then performs an update:

$$w \leftarrow (1 - \alpha) \cdot w + \alpha \cdot w_k, \quad \text{where } \alpha = \frac{1}{\sqrt{a - a_k + 1}}$$

$$a \leftarrow a + 1$$

    and returns the global model $w$ and age $a$ to client $k$

    $w_k \leftarrow w$

    $a_k \leftarrow a$

### 3.3.1 Restrictions on the age filter

While CO-OP introduces two additional parameters, namely $b_l$ and $b_u$, little guidance is provided as to how one should choose these values. Only the generous constraint $b_l < b_u$ was given in the original paper [29]. However, choosing these parameters arbitrarily with only this constraint in mind can cause deadlocks in worst-case scenarios. The deadlock we consider is the situation where all clients are, at some point, deemed overactive, rendering the algorithm unable to progress.

We identify two additional constraints that should be fulfilled to avoid deadlock: $b_l < K$ and $b_u \geq 2b_l$. If the first constraint, $b_l < K$, is unfulfilled, then we are in fact 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. If $b_l$ was bigger or equal to the number of clients, then there would not be enough clients to indirectly increment $a$, and $a - a_k < b_l$ would always

hold true beyond the initial state of $a_k$. The second constraint, $b_u \geq 2b_l$, says that 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.5.

An example can be helpful to give a better intuition of why leaving $b_u \geq 2b_l$ unfulfilled can cause a deadlock. Table 3.1 gives a CO-OP instance that deadlocks even though both other constraints are fulfilled. To see why this instance deadlocks, we inspect the state when **client 2 updated** the server, where we have the following inequalities:

$$\begin{aligned} a - a_1 &< b_l, \\ a - a_2 &< b_l, \\ a - a_3 = a &> b_u. \end{aligned}$$

These inequalities are interpreted as clients 1 and 2 being overactive, whereas client 3 is outdated. The outdated client 3 then reconciles its age with the server and thereby becomes overactive in the next state. Now, all three clients are overactive, leaving the algorithm in a state of deadlock. Note that another interleaving exists that does not cause the algorithm to deadlock: if client 3 reads $a = 3$ instead of $a = 4$, then client 3 will be allowed to upload and deadlock would not occur since $a = 5$ by the time of the final event.

**Table 3.1:** A state table for CO-OP with 3 clients, where $b_l = 2$, and $b_u = 3$. Note that the algorithm deadlocks after the third event.

| **Event** | $a$ | $a_1$ | $a_2$ | $a_3$ |
|---|---|---|---|---|
| initial state | 2 | 0 | 0 | 0 |
| client 1 updated | 3 | 3 | 0 | 0 |
| client 2 updated | 4 | 3 | 4 | 0 |
| client 3 outdated | 4 | 3 | 4 | 4 |
| clients overactive | 4 | 3 | 4 | 4 |

# 4

# Benchmark Design

There are many ways to benchmark machine learning algorithms. For instance, one has to decide what metrics, hyperparameter values, and network architectures to use in the given problem. These are all key decisions that will affect a benchmark's outcome. This chapter intends to make our benchmark design and methodology explicit.

## 4.1 The MNIST benchmark dataset

Our performance evaluation is based on how well each classifier performs digit recognition on images from the MNIST dataset [5]. MNIST consists of, in total, $70,000$ labelled images of handwritten digits. These are grayscale images, each with $28 \times 28$ pixels. The MNIST data are further divided into a test set comprising $10,000$ images and a training set of 60,000 images. MNIST is commonly used for benchmarking purposes, notably by both FedAvg [3] and CO-OP [29], which motivated us to also use it in our work.

## 4.2 Performance measures

Given the variety of available performance measures, the question of what measure, or possibly measures, we should use in our study naturally arises. This question is important because we cannot compare measurements of, say, accuracy with measurements of, for instance, precision or AUC. Because accuracy is the only consistently used measure in machine learning research, we also need to use this measure to make comparisons meaningful.

Although many studies use accuracy, it is good practice to consider if any other performance measures are particularly suitable or relevant for our study. We should expect different measures to rank the same classifiers differently [14], which makes finding a good measure a difficult task. Using accuracy merely out of convenience may therefore lead us to draw incorrect conclusions [14].

A common argument against accuracy is that class skew in the dataset makes accuracy biased toward the majority class. In such cases, the use of precision and recall is motivated since recall is a class-wise accuracy and precision will tell us how good our predictions are in each class. However, the dataset we consider is fairly balanced. The per-class distribution of MNIST is shown in Fig. 4.1, which shows a standard deviation of 5–6% relative to the mean. Therefore, class skew does not motivate us to use another metric.


**Figure 4.1:** MNIST class distribution in the training and test datasets. The line shows the average count. The standard deviations are approximately 322 and 59 for training and test datasets respectively.

A second argument against accuracy is that it assumes equal misclassification costs, that is, equal real-world costs of misclassification. This assumption might be problematic since it is often the case in practice that some classes are more important to correctly classify than others. The issue is sidestepped altogether if we can use a performance measure that is insensitive to misclassification costs, such as AUC.

Recall that AUC is only applicable to binary classification tasks. Because there is no machinery for multi-class ROC analysis [30], the available multi-class AUC measures are generalisations based on the binary measure. In light of the finding that AUC is incoherent with respect to different classifiers [25, 24], we are reluctant to use the multi-class variants based on an incoherent AUC measure. Since we are not aware of any multi-class variant of a coherent AUC, we will not consider any ROC-based measure.

We adopt the accuracy measure to quantify classification performance. Because class skew is not prevalent in MNIST and there is no suitable cost-insensitive multi-class measure, we find accuracy to be a reasonable choice and see no additional benefit from using precision and recall. We again note that the accuracy measure implicitly

assumes equal misclassification costs — no class is more important to identify than any other.

## 4.3 Evaluation approach

We follow the Bayesian approach recommended by [27] for our algorithm performance comparison. The Bayesian correlated t-test is employed since we only consider performance on a single dataset (MNIST). Tests are performed between all combinations of pairs of algorithms, resulting in $\binom{4}{2} = 6$ tests including the centralised approach. Each test results in a plot of a posterior distribution that describes the mean difference in accuracy between the tested classifiers. Since the posterior is a probability density function, we can use it to infer a probability of the hypothesis (the mean difference in accuracy) given the observed data. An analysis of these posteriors then constitutes our final performance evaluation.

As an example of how to interpret the results from a Bayesian correlated t-test, consider the posterior shown in Fig. 4.2. 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 (gives higher accuracies) 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, is 0.566, which we again interpret as a probability.

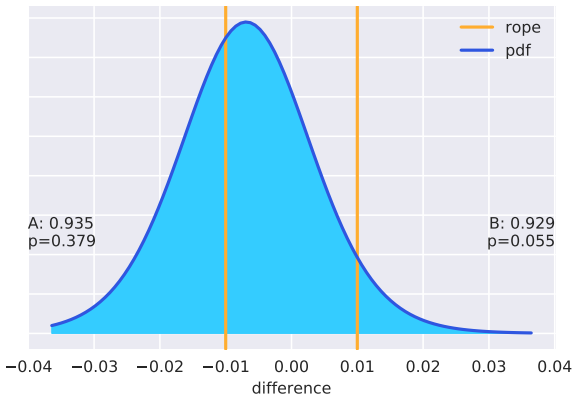



**Figure 4.2:** Posterior probability distribution of a correlated Bayesian t-test between classifiers $A$ and $B$. The vertical lines defines a region of practical equivalence where the mean difference in accuracy is no more than 1%.

One can argue that the two classifiers compared in Fig. 4.2 are practically equivalent because a majority of the distribution's mass lies within the rope. This is neverthe-

less an uncertain conclusion since 43.4% of the mass still lies outside of the rope. If the distribution instead had, say, 95% of its mass to the left of $-0.01$, then it may make sense to state that A is practically better than B. However, we could reverse the argument and say that A should be the preferred algorithm since the probability that B is practically better than A is only 5.5%.

Note that the analysis in the above example is heavily dependent on the choice of rope interval. In one extreme, a rope defined by the interval $(-\infty, \infty)$ would correspond to saying that all classifiers are practically equivalent, which is nonsensical. At the other extreme there would be no rope, which is to say that two classifiers never have the same performance even though a small difference might not be of practical importance. A reasonable definition is to say that the difference in accuracy has to exceed one percentage point for it to be practically significant, which implies the rope interval $[-0.01, 0.01]$. The interval can also be chosen based on domain-specific knowledge, for instance we can define a wider interval if a certain domain accepts more misclassifications. However, we use the $[-0.01, 0.01]$ interval in our comparisons.

We used the Python library *baycomp*[1] (provided by the authors of [27]) to perform Bayesian correlated t-tests. In baycomp, a posterior probability distribution is generated from a list of differences in accuracy of two classifiers from $i$ iterations of $k$-fold cross-validation. The input must come from cross-validation since the library performs a correction under this assumption. For us, this motivates a distributed partitioning of the MNIST data into folds, as described in 4.4. We chose to use 5-fold cross-validation because it gives us a decent compromise between additional computation and retained test examples (higher $k$ means more computation and smaller folds).

## 4.4 Data distribution

All of our distributed benchmarks contain a total of 100 clients and it is up to us to distribute the MNIST training data between these clients. A straightforward way to distribute a dataset is to shuffle the data and distribute an equal amount to all clients. This will give us Independently and Identically Distributed (IID) data; no client is special. Assuming IID data in a federated setting does not, however, reflect a realistic scenario and we will therefore also use non-IID data.

We use the term *non-IID 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 [3]. 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

[1] baycomp library: baycomp.readthedocs.io/en/latest/index.html

client had 2 shards of size 300. An example of how our IID and non-IID distributions compare to each other is given in Fig. 4.3.

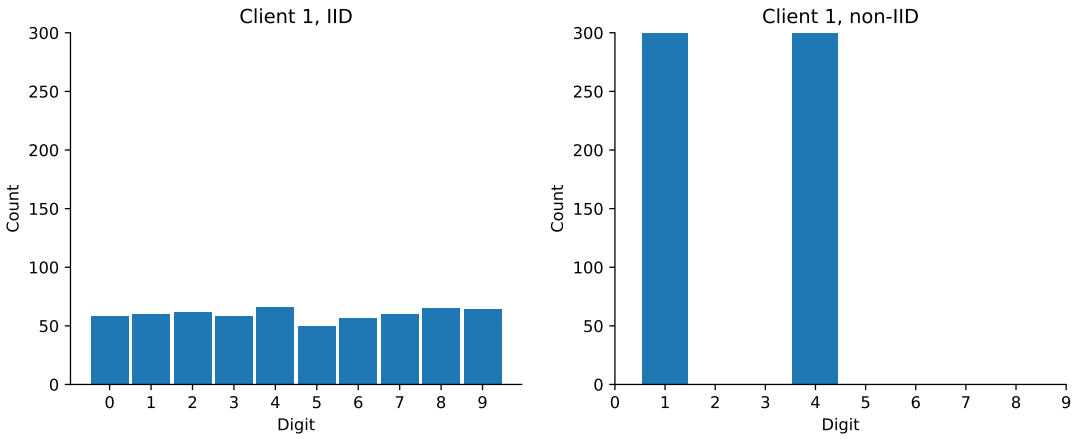



**Figure 4.3:** Side-by-side comparison of an IID and non-IID distribution from an actual client we used.

### 4.4.1 Versions for cross-validation

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

The first two steps of the non-IID approach applied to folds is the same as for the IID approach. The rest is almost the same as the previous non-IID approach. The step-by-step process is:

1. Shuffle the dataset.
2. Divide it into 5 folds.
3. Sort each fold.
4. Divide each fold into 200 shards of size 70.
5. Uniquely assign 2 random shard positions to each client.
6. Assign 2 shards to each client at its shard positions from each fold.

A difference from the previous non-IID approach is the decrease in shard size, from 300 to 70, but each client also receives more shards, from 2 to $5 \times 2$. However, this does not mean that each client is expected to have 5 times as many unique

labels. The shards that are taken from the same position in each fold are expected to contain the same labels. They are expected to be the same 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 still considered to be non-IID. In practice we saw that most clients had 2 unique labels, a few 3 or 1, and in rare cases one client had 4 unique labels.

## 4.5 Artificial neural network architecture

A multilayer perceptron with two hidden layers, referred to as *2NN*, is used in [3]. We used the 2NN architecture to obtain comparable results. The 2NN has two hidden layers, where each hidden layer has 200 neurons and each neuron has a ReLu activation function. Since we used this for MNIST it has $28 \times 28 = 784$ inputs and 10 outputs. The activation function for the output layer is a softmax function. Moreover, we use the cross-entropy loss function, which is a common choice for neural networks [26, Sec. 6.2]. In total, 2NN has $(784+1)\cdot200+(200+1)\cdot200+(200+1)\cdot10 = 199210$ trainable parameters.

## 4.6 Choosing hyperparameters

To have as fair a comparison as possible between the different optimisation algorithms, we need each algorithm to perform as well as possible. The performance of our algorithms is highly dependent on the configuration of their hyperparameters. Therefore, we should try to find the optimal set of hyperparameters for each algorithm.

H. B. McMahan, et al. did a hyperparameter search for FedAvg [3] and found that "the optimal learning rates do not vary too much as a function of the other parameters", but they did not present the optimal values. Because of this, we did parameter searches of our own for learning rate $\eta$ and learning rate decay $\lambda$ while keeping the other parameters fixed. We also explored how the algorithm behaves with different $C$ and $E$ values.

FSVRG and CO-OP have no explicit parameter search in their papers. The only adjustable hyperparameter for FSVRG is the *stepsize* $h$, but no explicit value is given in [4]. The same goes for CO-OP, where no values are given to $b_l$, or $b_u$ in [29]. CO-OP did, however, mention a batch size and a learning rate for their experimental setup, but not for the simulation setup that we were interested in. Because these previous works do not share their parameter values, we had to explore $h$, $b_l$, $b_u$, and $\eta$ together with a decay to find the best configuration for our particular problem.

Our approach to searching was either grid-based or randomised. Both approaches

require an interval to limit their search space. Grid search is performed by dividing the search space into a grid with a desired number of points. Random search is performed by picking random points in the search space, often with a uniform probability and a multidimensional search space [31].

Before we searched for a decay we searched for learning rates without a decay to see potential improvements with decay and the breaking point where $\eta$ is too big. The first learning rate values were $3^{-e}, e = 3, 4, ..., 8$. From those observations, we did a second search in the interval $(0.02, 0.09)$.

Usually, a learning rate decay is used when smaller steps are necessary after a while to hone in on the optimal value. There is, however, another benefit of using a decay alongside FedAvg. FedAvg does not always let all clients train at the same time, which means they will temporarily be out of sync. Some clients might contribute more to the global model than others, making the global model biased towards those clients and their data. A learning rate decay only decreases the learning rate for a client when it trains. Therefore, the global model is less biased towards clients that train often when a learning rate decay is used. The CO-OP algorithm also shares some of these benefits since some clients might make more updates than others. There is, however, a drawback for using a decay with CO-OP: a client who is outdated will discard an update which has already decayed its learning rate. If the learning rate is not explicitly reverted in this case, then the learning rate is decayed even though no progress was made.

We utilised random search when searching for a pair of learning rate and decay. Learning rates were taken uniformly from the interval $(0.02, 0.15)$ and learning rate decays from the logarithmic interval $(-8, -3)$ with base 10. The search space was sampled 20 times and used for FedAvg. A subset of these values were reused for CO-OP, which makes their results more comparable.

The stepsize $h$ can be interpreted as a learning rate, since in FSVRG a local model is updated with $h_k$ times a gradient. From what we observed from FedAvg, we wanted $h_k \times B$ to be between about 0.05 and 0.15. The factor $B$ is included because FSVRG will do $B$ times more updates compared to an algorithm using a batch size. The rearranged formula for $h_k$ is $h = h_k \cdot n_k$, where $n_k = 600$ with 100 clients. For FedAvg, we used a batch size $B = 20$, meaning we tried $h$ values between 1 and 5.

Hyperparameter optimisation is computationally expensive, especially if the search space is large. Therefore, we handpicked values which we were particularly interested in for a few parameters. More specifically, the handpicked values were for the parameters $C$ and $E$ from FedAvg, and the age filter from CO-OP. The $C$ and $E$ values were selected separately and then combined into a combinatorial grid search pattern. The bounds for the age filter were picked to at least conform to our constraints. We included extreme values like $b_l = 0$ and $b_u = \infty$. We also kept track of how many times a client was outdated and overactive to try to predict what would give a good performance.

We used the *Hyperas*[2] library to find parameters for the centralised learning approach. Hyperas was used to perform random search, for which we included parameters $\eta$, $\lambda$, and $B$.

All hyperparameter searches ran with MNIST non-IID, because out of our two distributions it is the most difficult one to learn. The parameter values and their corresponding results are reported in Sec. 6.1.

## 4.7 Termination criterion for cross-validation

Running the algorithms until they converge in our final comparison would take too much time on our experimental setup. Instead, we need a fair measurement that works for all of them, like the same total number of uploaded models from clients. The 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 10000 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 of the 100 clients will upload two times each round; one upload for the full gradient and one for the updated model.

## 4.8 Framework stress test

Ideally, we would demonstrate that the Erlang framework scales with the number of added nodes, that is to say, investigate if the framework could be used on a large vehicle fleet. However, this requires that each node has access to a separate processor core, and since we do not have 100 processors, our client nodes must compete for resources. Since the CPU is a shared resource between nodes, it is clear that our setup cannot demonstrate real-world scalability. The best we can do is to launch as many concurrent nodes as possible and have each node do a minimal amount of work, that is, immediately replying to the server. Such an experiment has been carried out to show that the framework can handle many simultaneously connected clients. Because the experiment results are not immediately relevant to our evaluation, we defer these results to Appendix A.4. The remainder of this section motivates how the scalability experiments were performed.

In the first experiment, only the Erlang distribution framework is considered, leaving out Python computations and communication via JSON. Including Python would introduce unpredictable overhead due to periodical thread blocking between checking the file system for updates. Such variability is generally undesirable since we

[2]Hyperas: maxpumperla.github.io/hyperas

follow [32] by measuring scalability as throughput — the total number of successful distributed Erlang operations — after a fixed amount of time.

Contrary to [32], we are limited to the processing power of three four-core machines. It is then to be expected that our first experiment quickly reaches a point where the processors become overwhelmed. The limitation in computing power can be artificially bypassed if we significantly increase the latency of each pong response. Additional latency is trivial to introduce without additional computational strain, for instance, by sleeping or blocking until a specific event occurs. In the second experiment, we achieve this by relaying the pong response through the computing framework, that is, including Python and communicating with JSON files. With additional latency, we observe linear scaling in the number of clients. Details are provided in Appendix A.4.

# 5

# Implementation

A distributed system is required to perform Federated Learning. This chapter contains an overview of how our system is implemented and how we used it to obtain results from running our algorithms.

## 5.1 Erlang distribution framework

Most work on Federated Learning focuses on the optimisation algorithms, while little to no time is spent considering the distribution framework. An exception is found in [33], where the authors demonstrate that the functional programming language Erlang is well suited to handle the distribution of computations to local nodes in the context of Federated Learning. For computation on the clients, they used either *Erlang* or *C*. However, we used *Python* together with the machine learning library Keras[1] for its ease of use.

We need some means of communication between Erlang, which handles the distribution of the model, and Python, which performs machine learning tasks. A simple approach is to read and write to JSON-files. A more direct approach to Erlang interoperability is via Erlang ports, where one can either write custom port drivers or use existing interfaces, for instance ErlPort[2] or Pylang [34]. Using ports will free us of overhead otherwise introduced by file I/O in the JSON approach. The main downside with ports is that each language we want to support requires custom port drivers, and writing such drivers is a more intricate task compared to parsing a JSON file. Since the potential speed-up of using ports would be invisible in our benchmarks, the straightforward JSON approach is used in our implementation.

Though Erlang excels at handling many concurrent processes (scaling within a node), it scales poorly in the number of distributed nodes. Here, a node refers to an *Erlang Virtual Machine* (Erlang VM) running on a host machine. Erlang's scaling issues across nodes are mainly due to *transitive connections* and *global name registration*, which are mechanisms for fault tolerance [35].

[1] Keras: github.com/keras-team/keras
[2] ErlPort: github.com/hdima/erlport.org

Using Erlang's transitive connections implies a fully connected network of nodes; if node $a$ connects to node $b$, then $a$ will also create connections to all of $b$'s neighbours and vice versa. The total number of connections is therefore quadratic in the number of nodes, $\mathcal{O}(n^2)$. Because these are live TCP/IP-based connections, simply maintaining all connections can strain the communication network. Previous benchmarks also show that the frequency of global operations is a severe bottleneck for scaling a fully connected network of nodes [32]. To deal with these issues in practice, a common ad hoc approach is to disable transitive connections [36, Sec. 3.2], for instance by using so-called hidden nodes that do not share their connections at the cost of reduced capabilities for fault tolerance. We apply this approach by turning all clients into hidden nodes that explicitly connects to the server, so that each client node is aware of no other connection than to the server.

To reflect a practical framework for Federated Learning we added a *user node*. The user node represents a data analyst who assigns machine learning tasks to the central server, which in turn distributes the computations to clients. The user node consists of an *Erlang node* that communicates with the server and a *Python script* for a programmer to communicate with the Erlang node. There could be several user nodes, but one user is sufficient for benchmarking purposes. The added benefit of using Erlang for communication is that we can easily move the user interactions to a computer separate from the server.

Nodes U, S, and C in Fig. 5.1 are written in Erlang. Node $C_i$, $i = 1, 2, ..., n$, communicates with *Client* $i$ through JSON, which is also used in communication between node U and *User*. In our case, both *User* and *Client* are written in Python. Since we apply a JSON-based approach, the user and client could be written in any language that can read and write JSON files, making the Erlang framework language-agnostic.

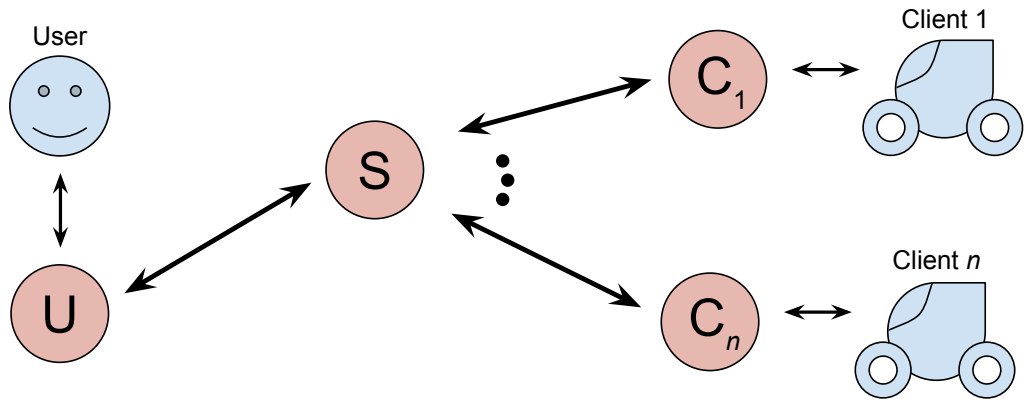



**Figure 5.1:** The nodes U, S, and C are Erlang nodes. User and Client are language-agnostic as they only need to communicate via JSON.

## 5.2 Client data

We distribute MNIST data statically to clients in multiple ways, as discussed in Sec. 4.4. These are the IID and non-IID partitionings as well as several versions of IID and non-IID divided into folds that are used in cross-validation. All distributions are static in the sense that we create them once and store them on the file system. In terms of implementation, each client is associated with unique files that contain raw training data and respective labels. Therefore, clients access their data from a particular partitioning by reading the correct files directly from the file system. Note that no training data is communicated over the Erlang framework in this approach, which reflects how Federated Learning would be performed in practice.

Different iterations of cross-validation should use different data in their folds. Because we perform several cross-validation iterations, we create a new partitioning (IID or non-IID) for each iteration.

## 5.3 Benchmarks

We primarily utilised the user node to run benchmarks. Since we can interact with the user node with Python, we can write a benchmark script that writes one or more JSON files and then awaits a result for each JSON file. Each JSON file is called an assignment, which contains information about what termination criterion, dataset, algorithm, and hyperparameters to use.

For FedAvg and FSVRG we evaluate the global model on a test set once per communication round. CO-OP, on the other hand, does not have an equally convenient opportunity to perform an evaluation. In CO-OP, evaluating the global model every time it is updated would slow down our benchmarks because a single update from a client is fast compared to a synchronous communication round. Therefore, we used an evaluation frequency of once every $10^{\text{th}}$ update. This is also comparable to FedAvg with $C = 0.1$ in the number of models uploaded per evaluation.

The server is written in Erlang which is not well suited for machine learning computations. Therefore, we implemented the evaluation itself in Python — the same way it was implemented on the clients. The communication between the server and the Python evaluation process is done with JSON just like it was described in Fig 5.1.

## 5.4 Experimental setup

During our benchmarking we mostly used three computers. Tab. 5.1 shows their specifications. Each computer could simulate 100 clients with at most around 40 clients training simultaneously before it ran out of memory. Even when three computers would give better CPU utilisation than one, it was still a good idea to run on as few computers as possible when we could. Running FedAvg with $C \leq 0.2$ was even faster on one computer, because of the overhead of sending data over a LAN. However, neither FSVRG, CO-OP, nor FedAvg with $C = 1.0$ could run on less than three of our computers.

**Table 5.1:** Technical specifications of computer A, B, and, C.

| Computers | A | B | C |
|---|---|---|---|
| VirtualBox | 5.2.4 | 5.2.4 | 5.2.8 |
| Ubuntu | 16.04 LTS | 16.04 LTS | 16.04 LTS |
| CPU | i7-6700K | i7-6700K | i7-7700K |
| Virtual RAM | 23.5 GiB | 23.5 GiB | 23.4 GiB |

Clients were distributed evenly to our three machines, where one machine also acts as the server and user. Note that CO-OP is affected by how clients are distributed to the machines in our experimental setup. In CO-OP, 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 CO-OP's age filter. This behaviour may in turn cause clients on other machines to be outdated. To ensure consistent behaviour in all CO-OP benchmarks, the machines always had the same number of simulated clients.

## 5.5 Model merging in FedAvg

As noted in Sec. 3.1, the global update step in the FedAvg algorithm is somewhat imprecisely stated. This ambiguity forced us to actively choose which one of the three identified interpretations to use in our benchmarks. To make an informed decision about what update to implement, some kind of initial evaluation had to be carried out. A single run for each variant is presented in Fig. 5.2.

In Fig. 5.2, variant 1 converges much faster than the other two. Both variant 0 and 2 are impeded by being weighted against older models, although these weightings contribute additional stability. Variant 2 is stable enough that it is difficult to see the difference between the raw values in Fig. 5.2 and a smoothed plot (provided in appendix A.2 Fig. A.1).

All three variants ran for 15 hours each on one machine, where some had time to make more communication rounds than others. None of the implementations were

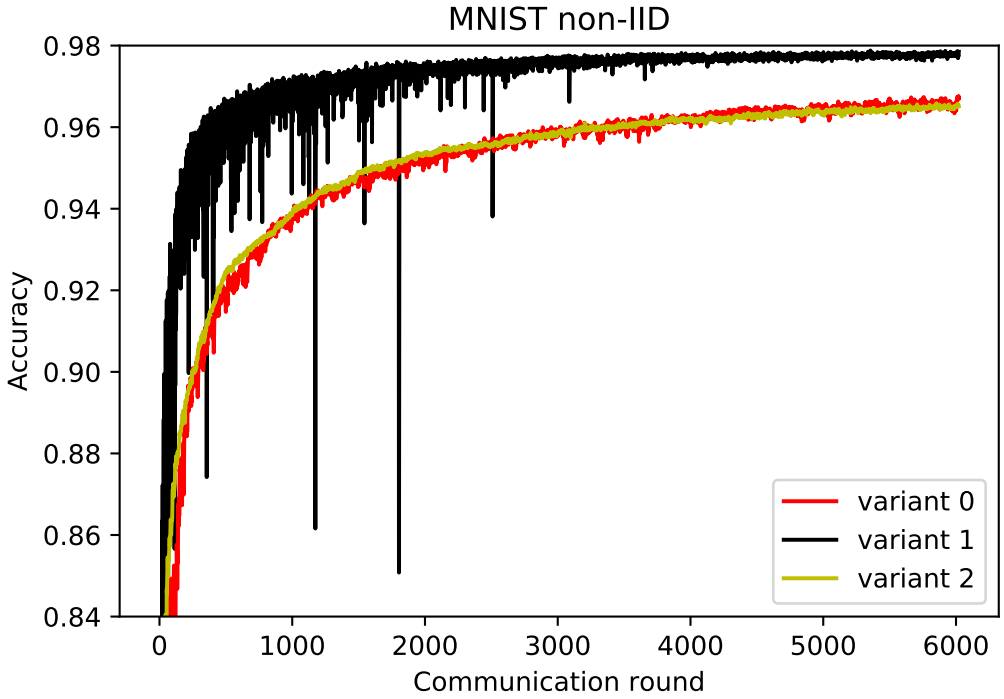



**Figure 5.2:** Three different variants of FedAvg. Variant 0, 1, and 2 represent the updates given by (3.1), (3.2), and (3.3), respectively. All of them used $C = 0.1$, $E = 5$, $B = 20$, $\eta = 0.05$, $\lambda = 10^{-5}$. Accuracy was evaluated after each communication round.

particularly optimised and they shared as much code as possible. With that said, the number of communication rounds in Fig. 5.2 is bounded by the slowest variant, which is variant 2. We conjecture that this is because a model has to be saved for each client, and those models will contribute to a sum that is computed every communication round.

We decided from this point onward to use variant 1. It converges faster and is presumably the variant that the original authors intended to convey. This hypothesis is supported by the fact that a follow-up paper [13] reformulates the global update to take the form of variant 1.

# 6

# Results

This chapter contains comparisons between FedAvg, CO-OP, FSVRG, and centralised learning. The first four sections are dedicated to finding good hyperparameter values for each algorithm and include the results of our hyperparameter searches. Finally, after deciding on what hyperparameters to use, Sec. 6.5 evaluates each algorithm via 5-fold cross-validation and includes comparisons based on Bayesian correlated t-tests.

## 6.1 Optimised FedAvg

For FedAvg we searched for good values of learning rate $\eta$, learning rate with decay $\lambda$, and client proportion $C$ with local epochs $E$. The results from the hyperparameter searches of $\eta$ and $\eta$ with $\lambda$ are presented in the form of tables to summarise 1200 communication rounds of training. Two abbreviated versions of these tables are presented below in Tab. 6.1 and 6.2, while the complete results can be found in Appendix A.3. The values presented for each parameter setup are the maximum accuracy, the minimum error, the average accuracy, and something we call the *drop sum*. The drop sum is described mathematically in (6.1) as the sum of the difference between the accuracy of one communication round to the next when the accuracy decreases.

$$\text{sum(drop)} = \sum_{c=2}^{1200} \max\left(0, acc_{c-1} - acc_c\right) \tag{6.1}$$

Note the low average accuracy for $\eta = 0.09$ in Tab. 6.1. Higher learning rates are usually faster, but at some point they will start to become more unstable and eventually diverge, which is what happened with $\eta = 0.09$.

In Tab. 6.2, the configuration $\eta = 0.11, \lambda = 2.2 \cdot 10^{-7}$ gave the best result. Also note that a low drop sum is not always a positive trait: the configuration $\eta = 0.14$, $\lambda = 7.8 \cdot 10^{-4}$ has a very low drop sum, but it also diverges fast which is reflected in the average accuracy. In other words, it had a low drop sum because the accuracy could not drop much lower.

**Table 6.1:** A truncated table of the learning rate searches for FedAvg. The best values are coloured green. The other parameters were $C = 0.1$, $E = 5$, $B = 20$, $\lambda = 0$, and non-IID MNIST.

| $\eta$ | max(acc) | min(err) | avg(acc) | sum(drop) |
|---|---|---|---|---|
| $3^{-8} \approx 0.00015$ | 0.9052 | 0.3448 | 0.8345 | 4.216 |
| $3^{-3} \approx 0.037$ | 0.9727 | 0.1107 | 0.9429 | 5.429 |
| 0.05 | 0.9738 | 0.1083 | 0.9481 | 4.936 |
| 0.06 | 0.9761 | 0.1053 | 0.9513 | 4.489 |
| 0.07 | 0.9759 | 0.1049 | 0.9518 | 4.908 |
| 0.09 | 0.9307 | 0.2296 | 0.2515 | 6.804 |

**Table 6.2:** Random search for good values of $\eta$ and $\lambda$ for FedAvg. The best values are coloured green. The other parameters were $C = 0.1$, $E = 5$, $B = 20$, $\lambda = 0$, and non-IID MNIST.

| $\eta$ | $\lambda$ | max(acc) | min(err) | avg(acc) | sum(drop) |
|---|---|---|---|---|---|
| 0.05 | 1.6e-06 | 0.9767 | 0.101 | 0.9582 | 5.014 |
| 0.068 | 2.8e-07 | 0.978 | 0.109 | 0.96 | 5.058 |
| 0.088 | 3.2e-06 | 0.9788 | 0.09794 | 0.9638 | 4.115 |
| 0.098 | 2.4e-08 | 0.979 | 0.1325 | 0.9562 | 7.108 |
| 0.11 | 2.2e-07 | 0.9792 | 0.09315 | 0.9658 | 3.444 |
| 0.14 | 7.8e-04 | 0.6962 | 0.8643 | 0.09896 | 0.8438 |

### 6.1.1 $B$ and $E$

Using the best $\eta$ and $\lambda$ from the previous search in Tab. 6.2, we continued by looking for better $C$ and $E$ values. Fig. 6.1 shows the results of this. Contrary to our previous beliefs from Sec. 4.6, our best $\eta$ and $\lambda$ values did not transfer well to other $C$ and $E$ values. All three $E$ values diverged with $C = 0.5$ and $C = 1.0$ diverged for the case of $E = 1$.

The divergent results from Fig. 6.1 made it difficult to observe the relationship between different $C$ and $E$ values. Therefore, we chose some other parameters to prevent divergence. In Fig. 6.2, we can see how $E = 1$ is slower than the others and how stable $C = 1.0$ is compared to the others. Another noteworthy observation from Fig. 6.2 compared to Fig. 6.1 is that $C = 0.2$ does not always have the highest maximum accuracy.

The final hyperparameter configuration we settled for was $\eta = 0.088$, $\lambda = 3.2 \cdot 10^{-6}$, $B = 20$, $E = 10$, and $C = 0.1$, which can be seen in Fig. 6.2b. We initially went for $\eta = 0.11$ and $\lambda = 2.2 \cdot 10^{-7}$. Unfortunately, those values were not only unstable for different $C$ values but also caused instability during cross-validation where less training data is used. A new pair, $\eta = 0.088$ and $\lambda = 3.2 \cdot 10^{-6}$, was then selected to have a less aggressive learning rate and still maintain a high average accuracy. The main motivation for not using $C = 0.2$, even though it was more stable, was

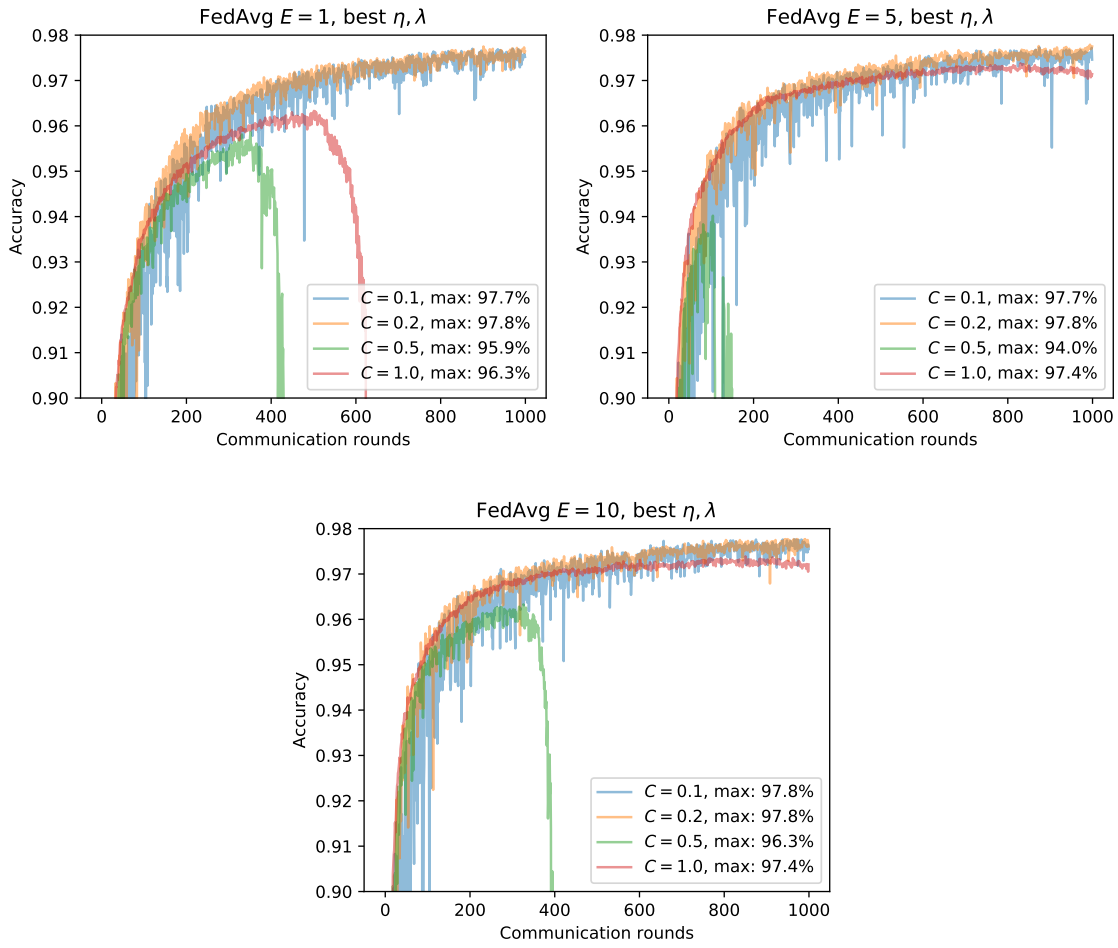



**Figure 6.1:** A single run of FedAvg with $\eta = 0.11$, $\lambda = 2.2 \cdot 10^{-7}$, $B = 20$ for different $C$ and $E$ values. The dataset was MNIST non-IID.

that $C = 0.1$ was much faster to run. Intuitively, $C = 0.2$ should be better because it has twice the number of updates, but the maximum accuracy was never really better. We applied similar reasoning to choose what $E$ value to use. The bigger the $E$, the slower the algorithm will be. From observations, $E = 5$ was not much slower than $E = 1$ while still converging faster. The parameter value $E = 10$ performed practically the same as $E = 5$, but was also noticeably slower and more unstable for $C = 0.1$.

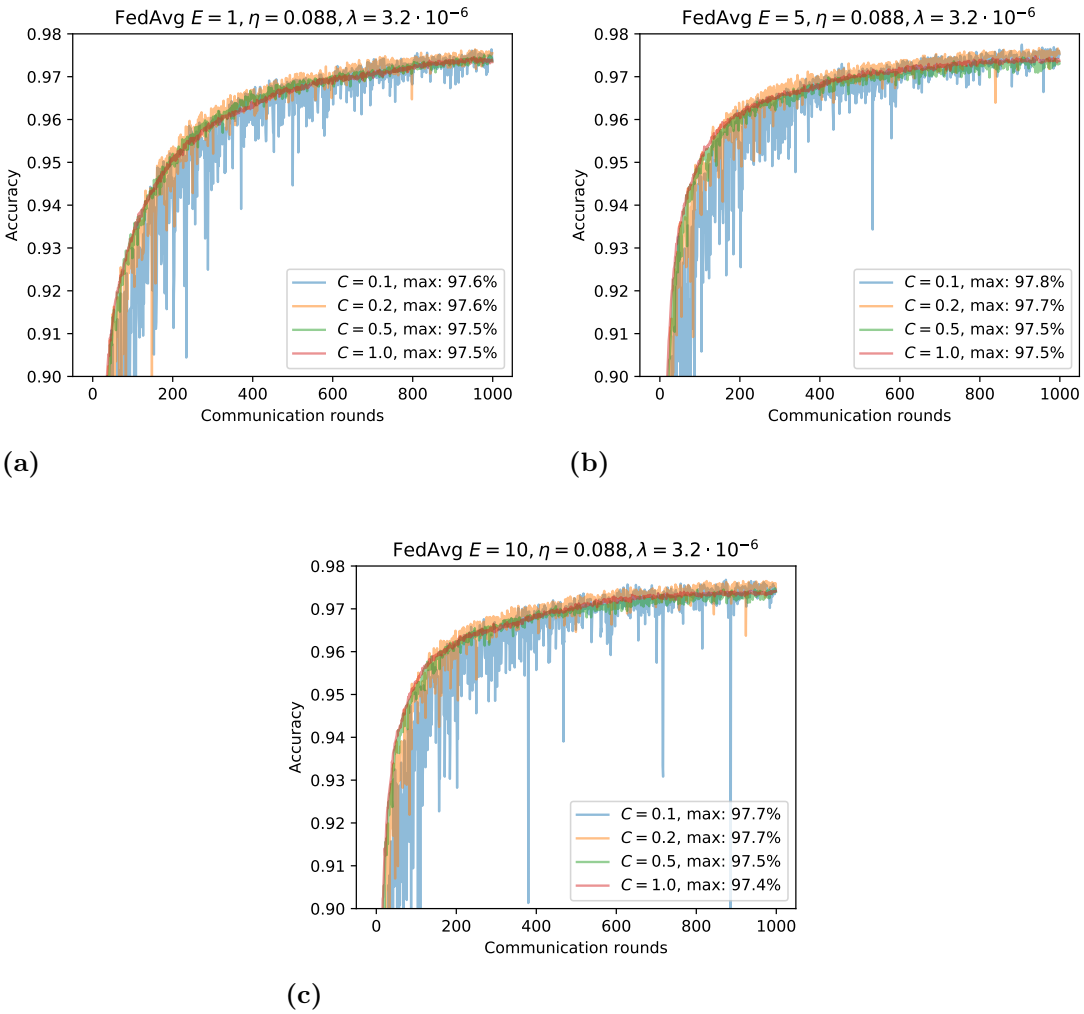



**Figure 6.2:** A single run of FedAvg with different $C$ and $E$ values; $B = 20$. The dataset was MNIST non-IID.

## 6.2 Optimised CO-OP

The subset we reused from FedAvg was the $\eta$ and $\lambda$ values that can be seen in Tab. 6.3. Interestingly enough, the best values from FedAvg ($C = 0.1$) also applied for CO-OP, but in this case it was stable enough to use. A graph representation with the same values can be seen in Fig. 6.3.

**Table 6.3:** Random search for good values of $\eta$ and $\lambda$ for CO-OP. The best values are coloured green. The other parameters were $b_l = 16$, $b_u = 51$, $E = 1$, and $B = 20$. The dataset was non-IID MNIST and all simulations ran until 50000 models were uploaded to the server.

| $\eta$ | $\lambda$ | max(acc) | min(err) | avg(acc) | sum(drop) |
|---|---|---|---|---|---|
| 0.05 | 1.6e-06 | 0.9566 | 0.1429 | 0.9106 | 23.28 |
| 0.061 | 1.8e-06 | 0.9613 | 0.1261 | 0.9186 | 21.52 |
| 0.068 | 2.8e-07 | 0.9632 | 0.1188 | 0.9218 | 21.75 |
| 0.081 | 3.2e-06 | 0.9661 | 0.1085 | 0.9273 | 19.49 |
| 0.088 | 3.2e-06 | 0.9665 | 0.1075 | 0.928 | 19.93 |
| 0.098 | 2.4e-08 | 0.9683 | 0.0989 | 0.9322 | 19.11 |
| 0.11 | 2.2e-07 | 0.9696 | 0.09602 | 0.9351 | 15.82 |
| 0.13 | 3.5e-04 | 0.9524 | 0.1618 | 0.9178 | 23.26 |
| 0.14 | 7.9e-05 | 0.9655 | 0.1121 | 0.9284 | 22.78 |

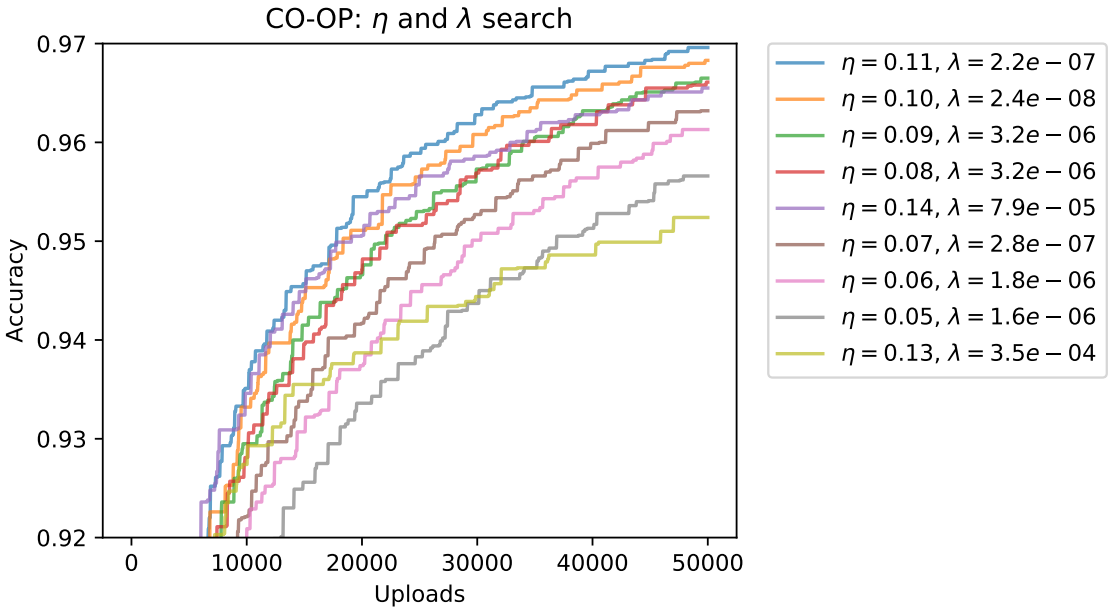



**Figure 6.3:** Monotonic representations of how CO-OP's training progressed for the different learning rates and decays from Tab 6.3. The legend is sorted from the highest to the lowest final accuracy.

The age filter used in these runs was arbitrarily chosen, within our restrictions from Sec. 3.3.1, to be $b_l = 16$ and $b_u = 51$. We also tried seven other filters and their performance was practically equivalent. The one with $b_l = 16$ was marginally better than the others while also having some outdated clients and some more overactive clients. The best values were chosen as our final configuration for CO-OP: $b_l = 16$, $b_u = 51$, $E = 1$, $B = 20$, $\eta = 0.11$, and $\lambda = 2.2 \cdot 10^{-}7$.

## 6.3 Optimised FSVRG

The search for stepsize $h$ started at 1 and was increased by 1 until the accuracy no longer improved. The best value we found was $h = 4$, which can be see in Tab. 6.4 and Fig. 6.4.

**Table 6.4:** A linear search for FSVRG's stepsize parameter $h$. The best values are coloured green. The dataset was non-IID MNIST.

| $h$ | max(acc) | min(err) | avg(acc) | sum(drop) |
|---|---|---|---|---|
| 1.0 | 0.975 | 0.07994 | 0.9314 | 1.031 |
| 2.0 | 0.9799 | 0.07058 | 0.9159 | 2.375 |
| 3.0 | 0.9801 | 0.06742 | 0.9157 | 2.29 |
| 4.0 | 0.9816 | 0.0663 | 0.9135 | 2.527 |
| 5.0 | 0.9811 | 0.06648 | 0.9081 | 3.769 |

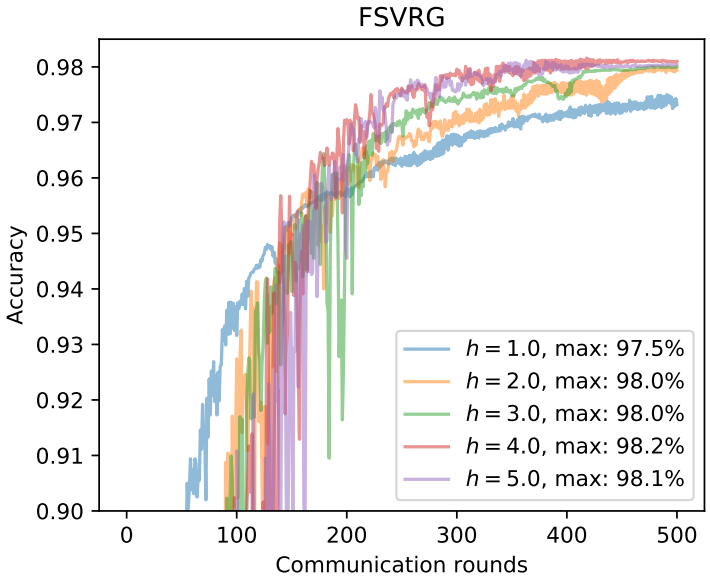



**Figure 6.4:** Graphical representation of Tab. 6.4, in which a linear search for a stepsize $h$ was performed for FSVRG.

## 6.4 Optimised centralised learning

The interval we used when searching for $\eta$ was between $10^e$, where $e \in (0, -2)$. For $\lambda$, the interval was $10^e$, where $e \in (-3, -6)$. And $B$ was uniformly taken from the set $\{10, 20, 50, 600\}$. The best parameter values found after testing 56 random configurations were $\eta = 7.03 \cdot 10^{-2}$, $\lambda = 3.77 \cdot 10^{-6}$, and $B = 10$. Each run performed at most 50 epochs.

## 6.5 Cross-validation and Bayesian analysis

Results from three iterations of cross-validation were used as input to each Bayesian correlated t-test. Many such tests were performed, and the resulting posterior distributions are shown in Fig. 6.5 (IID) and Fig. 6.6 (non-IID). The vertical lines show the region of practical equivalence, which we chose to be between $[-0.01, 0.01]$. Note that three of the comparisons in each figure are to centralised learning.

All probability density functions in Fig. 6.5 have more than 95 percent of their mass within one region. Because this mass is interpreted as a probability of relative performance, we immediately make the following decisions for IID data:

- FedAvg is practically better than both CO-OP and FSVRG.
- Centralised learning is practically better than both CO-OP and FSVRG.
- FedAvg and Centralised learning are practically equivalent
- CO-OP and FSVRG are practically equivalent.

Similarly, all density plots for comparisons on non-IID data in Fig. 6.6 have more than 95 percent of their mass within one region. Therefore, we decide on the following:

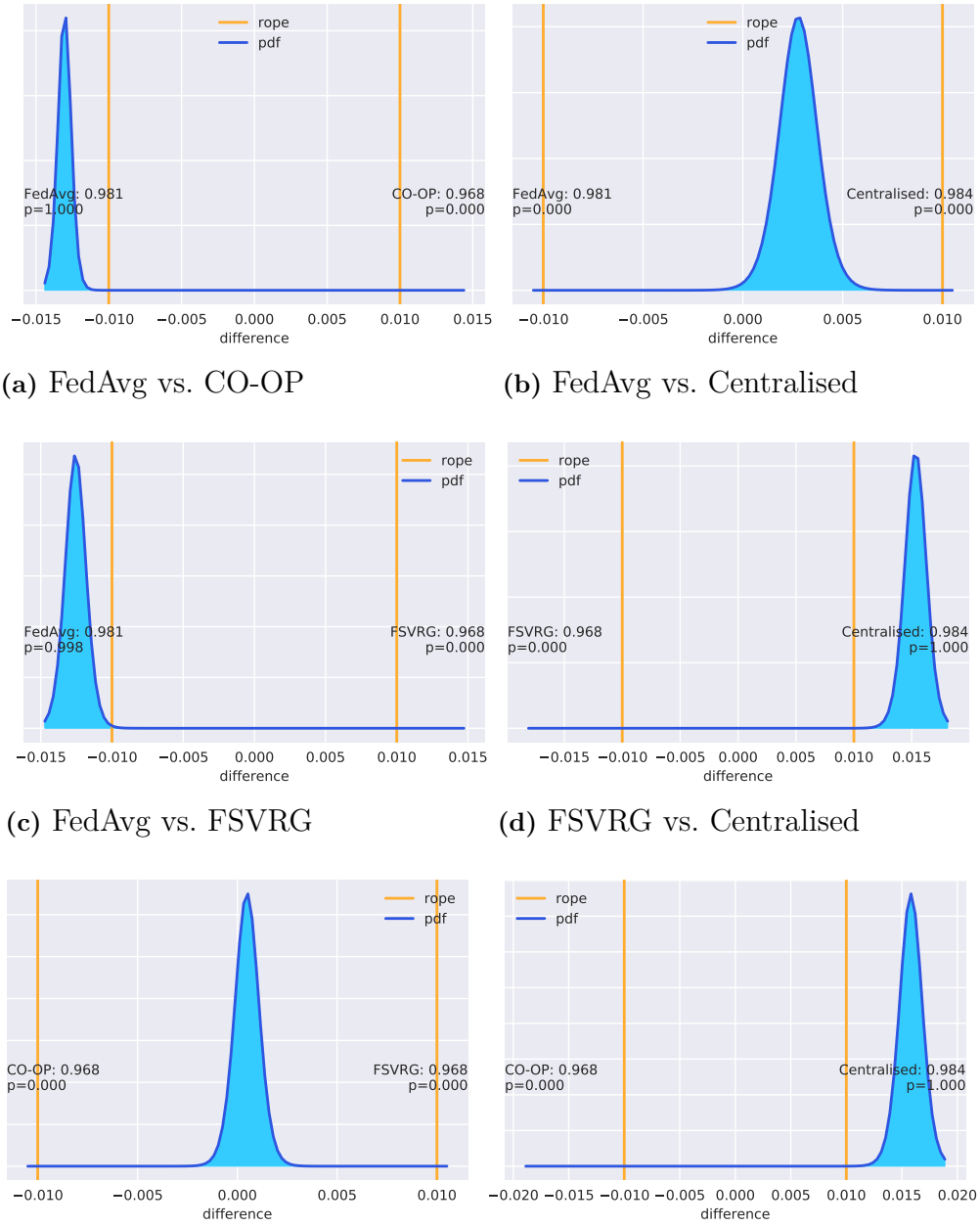
- FedAvg is practically better than both CO-OP and FSVRG.
- CO-OP is practically better than FSVRG.
- Centralised learning is practically better than FedAvg, CO-OP, and FSVRG.

The comparisons made so far might seem unfair since the federated algorithms are limited by the number of uploads and are therefore further away from converging. To give an extra edge to CO-OP and FSVRG, we let them do more uploads and refer to them as CO-OP$^{\times 5}$ and FSVRG$^{\times 10}$. These are shown in Fig. 6.7 and they have only performed two iterations of cross-validation. CO-OP$^{\times 5}$ did 5 times and FSVRG$^{\times 10}$ did 10 times more uploads than before. FSVRG$^{\times 10}$ made the greatest number of uploads, but it updated the global model with just as many local models as CO-OP$^{\times 5}$.

Note in Fig 6.7 that only the CO-OP$^{\times 5}$ vs. FSVRG$^{\times 10}$ comparison has more than 95 percent of its mass in one of the three regions. In the other five comparisons, we have to be more broad and say *practically equivalent or better*. We can then say the following from Fig. 6.7:

- FSVRG$^{\times 10}$ is practically equivalent or better than FedAvg
- FSVRG$^{\times 10}$ is practically better than CO-OP$^{\times 5}$

- FedAvg is practically equivalent or better than CO-OP$^{\times 5}$.
- Centralised learning is practically equivalent or better than FSVRG$^{\times 10}$.
- Centralised learning is practically better than CO-OP$^{\times 5}$.




**(a)** FedAvg vs. CO-OP

**(b)** FedAvg vs. Centralised

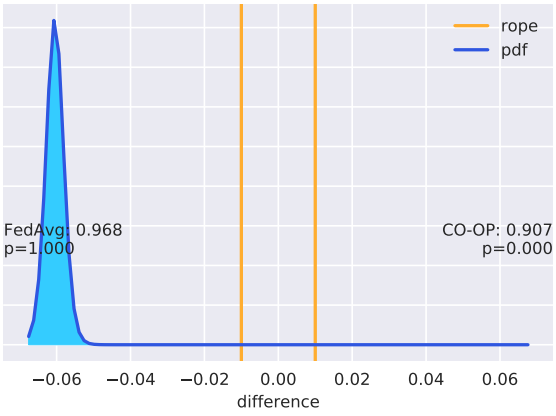
**(c)** FedAvg vs. FSVRG

**(d)** FSVRG vs. Centralised

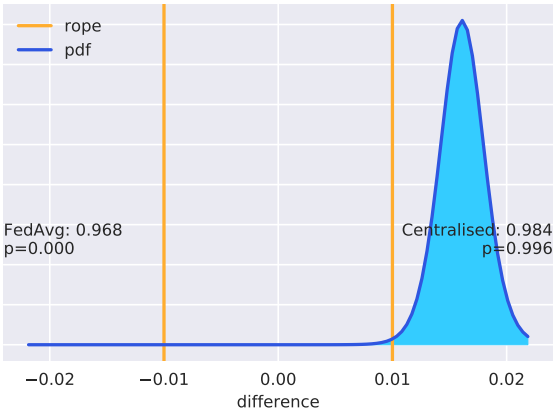
**(e)** CO-OP vs. FSVRG

**(f)** CO-OP vs. Centralised

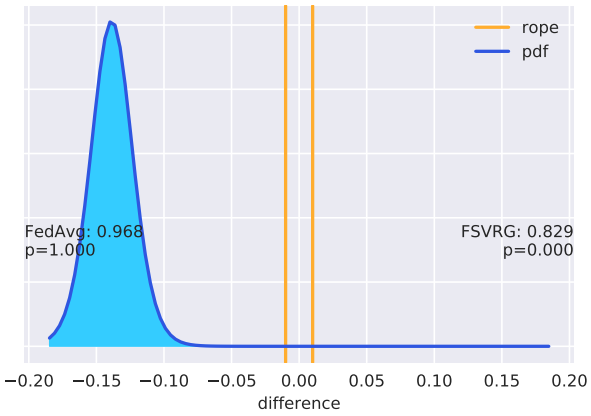
**Figure 6.5:** Comparisons between the federated optimisation algorithms on IID MNIST in form of posterior distributions. Each algorithm ran 3 iterations of 5-fold cross-validation. Each comparison has a mass $p$ on either side of the rope, which represents the probability of the algorithm on that side to be practically better. Note that the x-axes have different scales.



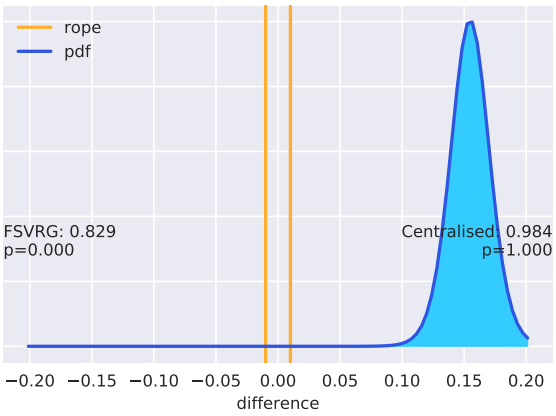

(a) FedAvg vs. CO-OP



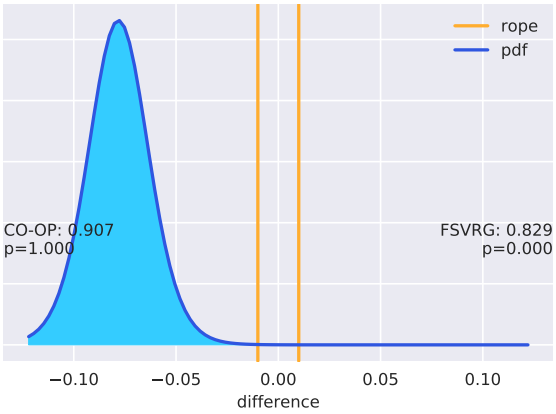

(b) FedAvg vs. Centralised



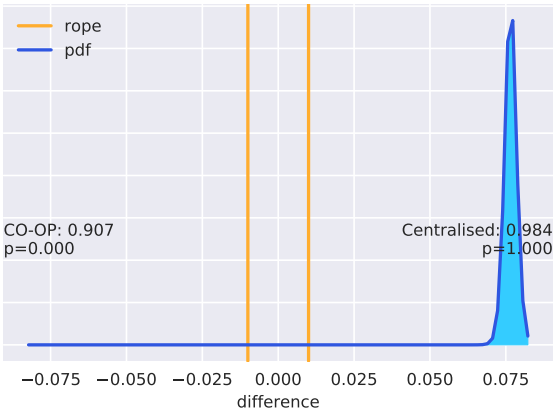

(c) FedAvg vs. FSVRG




(d) FSVRG vs. Centralised

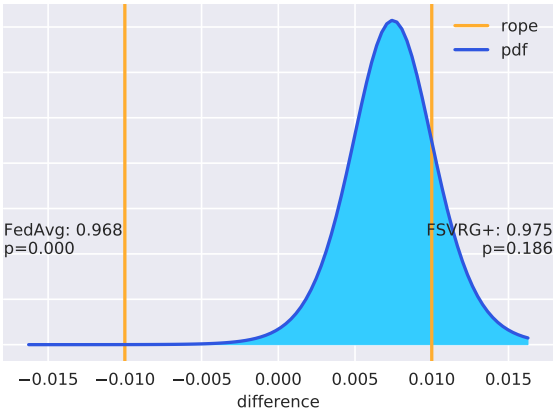



(e) CO-OP vs. FSVRG

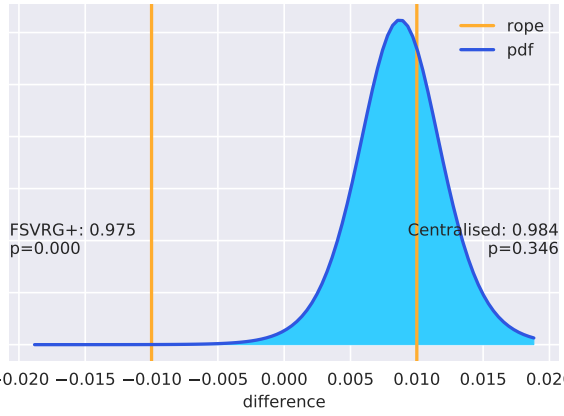



(f) CO-OP vs. Centralised

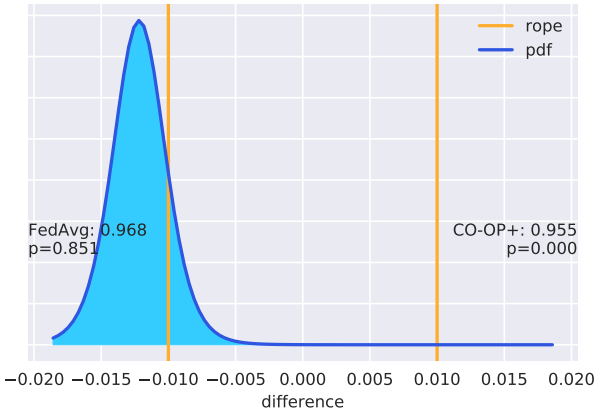
**Figure 6.6:** Comparisons between the federated optimisation algorithms on non-IID MNIST in form of posterior distributions. Each algorithm ran 3 iterations of 5-fold cross-validation. Each comparison has a mass $p$ on either side of the rope, which represents the probability of the algorithm on that side to be practically better. Note that the x-axes have different scales.



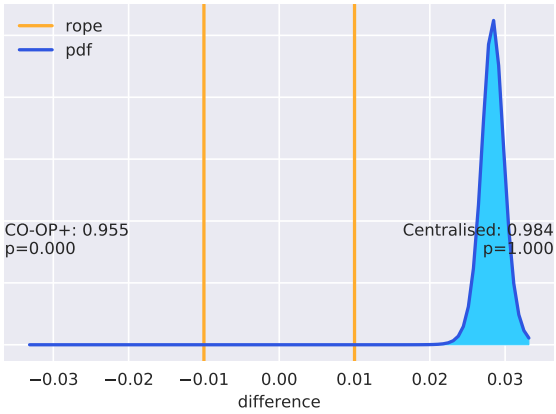

(a) FedAvg vs. FSVRG$^{(\times 10)}$



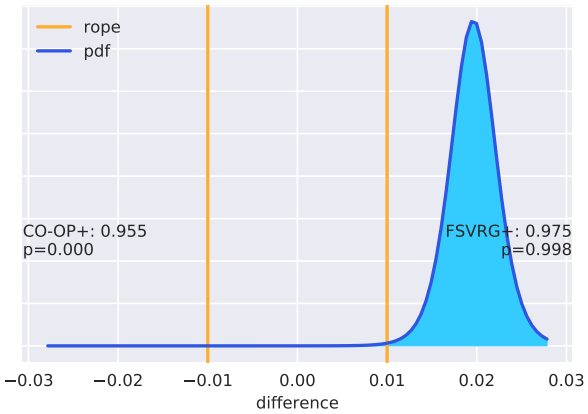

(b) FSVRG$^{(\times 10)}$ vs. centralised




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




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




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

**Figure 6.7:** Comparisons between the federated optimisation algorithms on non-IID MNIST in form of posterior distributions, where the superscripts indicate that a algorithm performed more uploads. Each algorithm ran 2 iterations of 5-fold cross-validation. Each comparison has a mass $p$ on either side of the rope, which represents the probability of the algorithm on that side to be practically better. Note that the x-axes have different scales.

# 7

# Conclusion and Discussion

In this thesis report we implemented and evaluated three Federated Learning algorithms (FedAvg, CO-OP, and FSVRG) on the MNIST dataset using 100 simulated clients. Our main goals were to evaluate if any algorithm is competitive with centralised learning and compare the algorithms to determine if any one of them should be the preferred choice. To this end, we also created a distributed framework for Federated Learning using the functional programming language Erlang. Moreover, we have explored three different approaches to model merging in FedAvg in Sec 5.5 and identified two new restrictions on the age filter used by CO-OP in Sec. 3.3.1.

## 7.1 Result discussion

Our main results consist of a performance comparison between the three algorithms as well as a comparison to a centralised approach. The results show that out of FedAvg, CO-OP, and FSVRG, FedAvg is practically better on MNIST digit recognition when we limit the number of uploads made by clients to 10000. When we let the others have an unfair advantage and do five times as many model uploads on non-IID MNIST, FedAvg is still competitive. FSVRG does, however, overtake FedAvg when it did ten times as many uploads. Moreover, even though CO-OP was the worst performing algorithm in terms of accuracy, it could still be viable in practice due to the advantage of being asynchronous.

The results also show that centralised learning is at least as good as FedAvg. This is not controversial since it is known that the centralised approach is easier to optimise. What is interesting, however, is that FedAvg was practically equivalent to centralised learning using IID MNIST. In particular, if we allow 1000 communication rounds, then FedAvg achieves an accuracy that is within one percentage point of a fully centralised approach on the IID partitioning. Moreover, given 1000 communication rounds, FedAvg is somewhere between one and two percentage points worse on the challenging non-IID partitioning. This shows that FedAvg has the potential to achieve good accuracies that are comparable to a centralised approach.

In our performance comparison, CO-OP sees no benefits in pure accuracy over FedAvg, not even if we allow CO-OP to perform five times as many normal uploads as

FedAvg. One difference between our and Wang's original CO-OP implementation is in how edge devices access their data. We assume that all data is available from the start, whereas Wang trains each client on a single batch per update to simulate a scenario where data is generated dynamically on edge devices. We have not explored if his approach is beneficial.

We were only able to perform 2–3 iterations of cross-validation for each algorithm. This number is quite low, and we would ideally have performed many more iterations. However, a complete iteration of cross-validation on all three algorithms takes about 25 hours (with 10000 uploads) or 90 hours (with more uploads) on our setup. Since our third machine was only sporadically available, completing tens of iterations was practically impossible. Note that including more iterations will give more representative result and is likely to somewhat change the appearance of the posterior plots. The probabilities of relative practical performance that we have presented should therefore not be taken completely at face value.

When performing cross-validation to obtain data for generating the Bayesian posteriors, we made use of the entire MNIST dataset (both test and training sets) when partitioning data into folds. These classifiers have therefore been allowed to *train* on parts of the original MNIST test set. This is not an issue in our comparison since all classifiers have trained on the same data distributions. However, note that the accuracies in Fig. 6.6, 6.7, and 6.5 should not be directly compared with results that are *evaluated* on the original MNIST test set.

## 7.2 Practical considerations

While FedAvg may be the best performing algorithm, CO-OP does enjoy the benefits of being asynchronous, which makes it worthwhile to consider in practice. An asynchronous algorithm does not have to wait for a set of clients to complete before model merging. Therefore, CO-OP is less sensitive to laggards and clients who loose their connection to the server. Also, CO-OP's age weighting scheme for model merging allows clients to dynamically generate new training data without having to communicate additional information, such as $n_k$. One of CO-OP's drawbacks is that the age filter bounds, $b_l$ and $b_u$, are difficult to tune and must be set with knowledge about the expected number of participating clients.

A straightforward observation is that FSVRG is strictly inferior to the other two in terms of communication efficiency. This is inherent to the FSVRG algorithm since every client is forced to upload its model gradient before any training can ensue. Therefore, FSVRG will always require more uploads per communication round compared to FedAvg. Because FedAvg also achieves better accuracies given same number of uploads, FSVRG has no practical benefits over FedAvg.

Federated Learning is not only motivated by big data, but requires a big data setting to be a sensible approach. If the amount of training data is not huge, then simply

uploading the raw data is quite possibly a more communication efficient approach. For instance, if we apply FedAvg to a dataset of 60MB (MNIST is $\approx$ 55MB), a model of 1.6MB ($200k$ 64-bit floats), and include 10 clients per communication round, then $10 * 1.6 = 16$MB would be uploaded by clients per communication round. This would only allow for $\approx 3.8$ communication rounds until clients have uploaded the same amount of data as the whole dataset. If consumer privacy laws do not apply to the intended application, then one should evaluate if there is enough data to motivate a federated approach or if a centralised solution would be more efficient.

While Federated Learning optimises for communication efficiency, our test environment makes several assumptions that are too strong in a deployed system. For instance, our hardware units are connected via stable Ethernet, we assume that messages arrive and are trustworthy, and we perform no encryption of transmitted data as suggested by [37, 38]. Also, our framework has a single point of failure, namely the server. In short, we have committed to several of the eight fallacies of distributed computing [39]. However, this is to be expected since our framework implementation is a research prototype rather than a production-ready system.

## 7.3 Related work

The term Federated Learning was coined by [3], who used their proposed algorithm FedAvg to evaluate the difference in the number of required communication rounds until convergence compared to synchronised stochastic gradient descent. Further work used compressed updates to reduce the total amount of communication data by two orders of magnitude, and still produce the same high quality model, but at the cost of additional communication rounds before convergence [13].

The FedAvg algorithm was preceded by [4], which introduced the first federated optimisation algorithm, FSVRG. Their work focused on analysis and experiments for convex optimisation. However, the authors also note that there is nothing stopping us from applying the non-federated version (SVRG) to non-convex problems, such as deep neural networks. Indeed, results from [40] suggest that SVRG is a viable alternative to SGD, though we are not aware of any previous attempts to evaluate FSVRG applied to non-convex optimisation. Hence, to the best of our knowledge, our FSVRG implementation is the first attempt to do so.

Wang's work on CO-OP [29] includes a simulation that compares CO-OP with FedAvg on MNIST digit recognition. The results suggest that CO-OP outperforms FedAvg in the number of rounds needed before convergence as well as converging to higher accuracies. However, Wang's FedAvg implementation performs much worse on MNIST (below 90% accuracy) compared to the original results in [3] (at least 97% accuracy). We see several possible reasons for this discrepancy. In Wang's comparison, FedAvg has been adapted to follow an asynchronous protocol and does

not strictly follow the original description since it does not, for instance, randomly select a subset of clients in each simulation round. We, on the other hand, leverage our distribution framework to separately implement synchronous FedAvg and asynchronous CO-OP without adapting one to the other. Moreover, Wang use a single layer ANN architecture with the *tanh* activation function. We adopt the 2NN architecture with ReLu as activation function used in [3] for a better comparison against their results.

We are aware of a few other Federated Learning algorithms besides the three we have implemented. These are MOCHA [41] and Federated Meta-Learning [42]. However, Federated Meta-Learning was only recently published and focus mainly on recommendation systems. The MOCHA algorithm formulates a federated version of multi-task learning, but is not applicable to non-convex deep learning.

## 7.4 Future work

CO-OP does not suffer as much from the practical issue of laggards as synchronous algorithms, such as FedAvg, do. Therefore, we can conjecture that CO-OP would in practice perform more updates than FedAvg during the same amount of time if an appropriate age filter is used. The question is then how much faster CO-OP is compared to FedAvg; is it just enough to compensate for the extra number of uploads needed to compete with FedAvg, or could CO-OP give better accuracies faster? While our results could suggest that CO-OP would have to upload more than five times as much as FedAvg, our simulated system is insufficient to answer this question.

One major alteration we made to our CO-OP implementation was how clients collect and train on data. The original paper aggregated a batch of data for a client before it could do an update [29]. We, on the other hand, let each client have access to all of its training data at all times. The natural question would then be: does the choice of method for aggregating data matter?

Because we only benchmark on MNIST, we can only draw conclusions about performance on that particular task. A more general result could be obtained if more datasets were included in the comparison. For instance, a Bayesian hierarchical t-test [27] would be able to summarise algorithm performance on multiple datasets.

In our work, we only considered a balanced distribution where all clients are given the same amount of data. However, this does not adequately reflect a federated setting where clients have different amounts of local data. While an uneven partitioning of data to clients is reported to slightly facilitate learning with FedAvg [3], it is unclear if the same is true for CO-OP and FSVRG. Investigating how CO-OP and possibly also FSVRG performs when client data is unevenly distributed would be an interesting continuation of our work.

When trying different age filters, we quickly discovered two possibilities of deadlocking for CO-OP. To avoid these deadlocks, we added two restrictions to the age filter (see Sec. 3.3.1). This indicates an immaturity in the theory behind CO-OP and a better understanding of the algorithm should therefore be explored. For instance, the restriction $b_u \geq 2b_l$ is not needed after the first $b_l$ updates. Setting the initial age values for the clients to something like $a_1 = 0, a_2 = 1, ..., a_K = K - 1$ may then make that restriction obsolete. Or perhaps a dynamically optimised age filter could remove the problem altogether.

# Bibliography


[1] M. Johanson, S. Belenki, J. Jalminger, M. Fant, and M. Gjertz, "Big automotive data: Leveraging large volumes of data for knowledge-driven product development," in *IEEE Int. Conf. Big Data (Big Data)*, Oct 2014, pp. 736–741. [Cited on pages 1 and 6.]

[2] Hitachi Data Systems, "The internet on wheels and Hitachi, Ltd." White Paper, Dec. 2015. [Cited on pages 1 and 6.]

[3] H. B. McMahan, E. Moore, D. Ramage, S. Hampson *et al.*, "Communication-efficient learning of deep networks from decentralized data," *arXiv preprint arXiv:1602.05629*, 2016. [Cited on pages 1, 2, 13, 19, 22, 24, 47, and 48.]

[4] J. Konecný, H. B. McMahan, D. Ramage, and P. Richtárik, "Federated optimization: Distributed machine learning for on-device intelligence," *arXiv preprint arXiv:1610.02527*, 2016. [Cited on pages 2, 15, 24, and 47.]

[5] Y. LeCun and C. Cortes, "The mnist database of handwritten digits," 1998. [Online]. Available: http://yann.lecun.com/exdb/mnist/ [Cited on pages 3 and 19.]

[6] T. M. Mitchell, *Machine Learning*, 1st ed. New York, NY, USA: McGraw-Hill, Inc., 1997. [Cited on page 5.]

[7] L. Deng, G. Hinton, and B. Kingsbury, "New types of deep neural network learning for speech recognition and related applications: an overview," in *IEEE Int. Conf. Acoust., Speech and Signal Process.*, May 2013, pp. 8599–8603. [Cited on page 5.]

[8] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, pp. 533–536, 1986. [Online]. Available: http://doi.org/10.1038/323533a0 [Cited on page 5.]

[9] D. Martín-Sacristán, J. F. Monserrat, J. Cabrejas-Peñuelas, D. Calabuig, S. Garrigas, and N. Cardona, "On the way towards fourth-generation mobile: 3GPP LTE and LTE-Advanced," *EURASIP J. Wireless Commun. and Networking*, vol. 2009, no. 1, Aug 2009. [Online]. Available: https://doi.org/10.1155/2009/354089 [Cited on page 6.]

[10] J. Hyun, Y. Won, E. Kim, J. H. Yoo, and J. W. K. Hong, "Is LTE-Advanced really advanced?" in *IEEE/IFIP Network Operations and Manage. Symp. (NOMS)*, April 2016, pp. 703–707. [Online]. Available: https://doi.org/10.1109/NOMS.2016.7502881 [Cited on page 6.]

[11] European Commission, "What data can we process and under which conditions?" Accessed on: Apr. 26, 2018. [Online]. Available: 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 [Cited on pages 6 and 8.]

[12] "Consumer data privacy in a networked world: A framework for protecting privacy and promoting innovation in the global digital economy," *J. Privacy and Confidentiality*, vol. 4, no. 2, p. 95–142, 2013. [Online]. Available: http://repository.cmu.edu/jpc/vol4/iss2/5 [Cited on page 6.]

[13] J. Konečnỳ, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," *arXiv preprint arXiv:1610.05492*, 2016. [Cited on pages 7, 33, and 47.]

[14] D. J. Hand, "Assessing the performance of classification methods," *Int. Statistical Rev.*, vol. 80, no. 3, pp. 400–414, 2012. [Online]. Available: http://dx.doi.org/10.1111/j.1751-5823.2012.00183.x [Cited on pages 8 and 19.]

[15] M. Sokolova and G. Lapalme, "A systematic analysis of performance measures for classification tasks," *Inform. Process. & Manage.*, vol. 45, no. 4, pp. 427 – 437, 2009. [Online]. Available: https://doi.org/10.1016/j.ipm.2009.03.002 [Cited on pages 8 and 9.]

[16] C. Ferri, J. Hernández-Orallo, and R. Modroiu, "An experimental comparison of performance measures for classification," *Pattern Recognition Lett.*, vol. 30, no. 1, pp. 27 – 38, 2009. [Online]. Available: https://doi.org/10.1016/j.patrec.2008.08.010 [Cited on page 8.]

[17] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *J. Mach. Learning Res.*, vol. 7, pp. 1–30, Dec. 2006. [Online]. Available: http://dl.acm.org/citation.cfm?id=1248547.1248548 [Cited on page 8.]

[18] F. J. Provost, T. Fawcett, and R. Kohavi, "The case against accuracy estimation for comparing induction algorithms," in *Proc. 15th Int. Conf. Mach. Learning (ICML)*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1998, pp. 445–453. [Online]. Available: http://dl.acm.org/citation.cfm?id=645527.657469 [Cited on page 8.]

[19] F. Provost and T. Fawcett, "Robust classification for imprecise environments," *Mach. Learning*, vol. 42, no. 3, pp. 203–231, Mar. 2001. [Online]. Available: https://doi.org/10.1023/A:1007601015854 [Cited on page 8.]

[20] T. Fawcett, "An introduction to ROC analysis," *Pattern Recognition Lett.*, vol. 27, no. 8, pp. 861 – 874, 2006. [Online]. Available: https://doi.org/10.1016/j.patrec.2005.10.010 [Cited on page 10.]

[21] D. J. Hand and R. J. Till, "A simple generalisation of the area under the roc curve for multiple class classification problems," *Mach. Learning*, vol. 45, no. 2, pp. 171–186, Nov. 2001. [Online]. Available: https://doi.org/10.1023/A:1010920819831 [Cited on page 10.]

[22] J. Huang and C. X. Ling, "Using AUC and accuracy in evaluating learning algorithms," *IEEE Trans. Knowl. Data Eng.*, vol. 17, no. 3, pp. 299–310, Mar. 2005. [Online]. Available: https://doi.org/10.1109/TKDE.2005.50 [Cited on page 10.]

[23] D. J. Hand and C. Anagnostopoulos, "When is the area under the receiver operating characteristic curve an appropriate measure of classifier performance?" *Pattern Recognition Lett.*, vol. 34, no. 5, pp. 492 – 495, 2013. [Online]. Available: https://doi.org/10.1016/j.patrec.2012.12.004 [Cited on page 10.]

[24] ——, "A better beta for the H measure of classification performance," *Pattern Recognition Lett.*, vol. 40, pp. 41 – 46, 2014. [Online]. Available: https://doi.org/10.1016/j.patrec.2013.12.011 [Cited on pages 10 and 20.]

[25] D. J. Hand, "Measuring classifier performance: a coherent alternative to the area under the roc curve," *Machine Learning*, vol. 77, no. 1, pp. 103–123, Oct. 2009. [Online]. Available: https://doi.org/10.1007/s10994-009-5119-5 [Cited on pages 10 and 20.]

[26] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org. [Cited on pages 11 and 24.]

[27] A. Benavoli, G. Corani, J. Demšar, and M. Zaffalon, "Time for a change: a tutorial for comparing multiple classifiers through bayesian analysis," *Journal of Machine Learning Research*, vol. 18, no. 77, pp. 1–36, 2017. [Online]. Available: http://jmlr.org/papers/v18/16-305.html [Cited on pages 12, 21, 22, and 48.]

[28] G. Corani and A. Benavoli, "A bayesian approach for comparing cross-validated algorithms on multiple data sets," *Mach. Learning*, vol. 100, no. 2, pp. 285–304, Sep 2015. [Online]. Available: https://doi.org/10.1007/s10994-015-5486-z [Cited on page 12.]

[29] Y. Wang, "CO-OP: Cooperative machine learning from mobile devices," Master's thesis, Dept. Elect. and Comput. Eng., Univ. Alberta, Edmonton, Canada, 2017. [Cited on pages 14, 16, 17, 19, 24, 47, and 48.]

[30] N. Lachiche and P. Flach, "Improving accuracy and cost of two-class and multi-class probabilistic classifiers using ROC curves," in *Proc. 20th Int. Conf.*

*Mach. Learning (ICML)*. AAAI Press, 2003, pp. 416–423. [Online]. Available: http://dl.acm.org/citation.cfm?id=3041838.3041891 [Cited on page 20.]

[31] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *J. Mach. Learn. Res.*, vol. 13, no. 1, pp. 281–305, Feb. 2012. [Online]. Available: http://dl.acm.org/citation.cfm?id=2503308.2188395 [Cited on page 25.]

[32] A. Ghaffari, "Investigating the scalability limits of distributed erlang," in *Proc. 13th ACM SIGPLAN Workshop Erlang*, 2014, pp. 43–49. [Online]. Available: http://doi.acm.org/10.1145/2633448.2633449 [Cited on pages 27 and 30.]

[33] G. Ulm, E. Gustavsson, and M. Jirstrand, "Functional federated learning in erlang," submitted for publication. [Cited on page 29.]

[34] R. Huang, H. Masuhara, and T. Aotani, "Pyrlang: A high performance Erlang virtual machine based on RPython," in *Companion Proc. 2015 ACM SIGPLAN Int. Conf. Syst., Programming, Languages and Appl.: Software for Humanity*, 2015, pp. 48–49. [Online]. Available: http://doi.acm.org/10.1145/2814189.2817267 [Cited on page 29.]

[35] N. Chechina, H. Li, A. Ghaffari, S. Thompson, and P. Trinder, "Improving the network scalability of Erlang," *J. of Parallel and Distributed Computing*, vol. 90-91, pp. 22 – 34, 2016. [Online]. Available: https://doi.org/10.1016/j.jpdc.2016.01.002 [Cited on page 29.]

[36] N. Chechina *et al.*, "Evaluating scalable distributed Erlang for scalability and reliability," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 8, pp. 2244–2257, Aug 2017. [Online]. Available: https://doi.org/10.1109/TPDS.2017.2654246 [Cited on page 30.]

[37] K. Bonawitz *et al.*, "Practical secure aggregation for federated learning on user-held data," *arXiv preprint arXiv:1611.04482*, 2016. [Cited on page 47.]

[38] L. T. Phong, Y. Aono, T. Hayashi, L. Wang, and S. Moriai, "Privacy-preserving deep learning: Revisited and enhanced," in *Appl. and Techn. Inform. Security*, L. Batten, D. S. Kim, X. Zhang, and G. Li, Eds., 2017, pp. 100–110. [Online]. Available: https://doi.org/10.1007/978-981-10-5421-1_9 [Cited on page 47.]

[39] A. Rotem-Gal-Oz, "Fallacies of distributed computing explained," White Paper, 2006. [Online]. Available: http://www.rgoarchitects.com/Files/fallacies.pdf [Cited on page 47.]

[40] S. J. Reddi, A. Hefny, S. Sra, B. Póczós, and A. Smola, "Stochastic variance reduction for nonconvex optimization," in *Proc. 33rd Int. Conf. Mach. Learning (ICML)*, vol. 48, 2016, pp. 314–323. [Online]. Available: http://dl.acm.org/citation.cfm?id=3045390.3045425 [Cited on page 47.]

[41] V. Smith, C. Chiang, M. Sanjabi, and A. Talwalkar, "Federated multi-task learning," *arXiv preprint arXiv:1705.10467*, 2017. [Cited on page 48.]

[42] F. Chen, Z. Dong, Z. Li, and X. He, "Federated meta-learning for recommendation," *arXiv preprint arXiv:1802.07876*, 2018. [Cited on page 48.]

# Acronyms

**ANN** Artificial Neural Network. 1, 3, 5–7, 9, 48, 59

**AUC** Area Under the Curve. 10, 19, 20

**AUROC** Area Under the Receiver Operating Characteristic curve. 10

**FCC** Fraunhofer-Chalmers Research Centre for Industrial Mathematics. 3

**FFI** Fordonsstrategisk Forskning och Innovation. 3

**FSVRG** Federated Stochastic Variance Reduced Gradient. v, 13, 15, 16, 26, 40, 41, 45–48

**IID** Independently and Identically Distributed. v, 2, 7, 22, 23, 31, 36, 39–45

**LAN** Local Area Network. 2, 32

**ROC** Receiver Operating Characteristic curve. 10, 20

**rope** region of practical equivalence. 12, 21, 22, 41–44

**SGD** Stochastic Gradient Decent. 5, 6, 13, 16

# Glossary

**2NN** is an ANN with 2 hidden layer and 200 neurons per layer (see Sec. 4.5). 24, 48

**federated setting** is a context where data is generated by many computers and they are connected to some kind of network (e.g. Internet of Things). 2, 7, 22, 48

**hyperparameter** within the area of machine learning is a parameter which is set before training. 13, 24

**Keras** is a machine learning library designed to be more intuitive to use, while using Tensorflow as a backend. 29

**learning rate** used in machine learning as a step size or a scaling factor to the gradient used for optimising a model. 13, 16, 24

**learning rate decay** is a way of decreasing learning rate after each epoch. Intuitively, it is a way of learning a lot in the beginning and less in the end when there are only finer details left to learn. 13, 16, 24, 25

**multilayer perceptron** is a subset of feedforward artificial neural networks with multiple fully connected layers. Each neuron also has a non-linear activation function. 24

**ReLu** stands for rectified linear unit and is a common activation function for ANN. The vanilla version is defined as $f(x) = \max(0, x)$. 24, 48

**softmax function** or normalised exponential function, is a logistic function that makes a multidimensional vector to sum to 1. Commonly used for classification tasks to represent a probability distribution in the output. 9, 24

# A

# Appendix 1

## A.1 Notation

**Common notation**

| | |
|---|---|
| $K$ | number of clients |
| $n$ | total amount of data (training examples) over all clients |
| $(x_i, y_i)$ | i:th training example |
| $\mathcal{P}_k$ | set of indices for the training examples residing locally on client $k$ |
| $n_k = \lvert\mathcal{P}_k\rvert$ | the amount of data residing locally on client $k$ |
| $w_t$ | global model at timestep $t$ |
| $w_t^k$ | client $k$'s locally trained model at timestep $t$ |
| $d$ | dimensionallity of $w_t$ and $w_t^k$ |
| $\eta$ | learning rate |
| $\lambda$ | learning rate decay |
| $\mathcal{B}$ | |
| $B$ | batch size used in SGD |
| $E$ | the number of epochs before a client sends an update |

**Specific to Federated Averaging**

| | |
|---|---|
| $C$ | fraction of clients that will participate in a communication round |
| $S_t$ | set of clients that participate in the communication round at timestep $t$ |
| $\mu = \sum_{k \in S_t} n_k$ | total amount of data (training examples) over all clients in $S_t$ |
| $\lvert S_t\rvert = C \cdot K$ | number of participating clients in the communication round at timestep $t$ |

**Specific to CO-OP**

| | |
|---|---|
| $a$ | age of the shared global model |
| $a_k$ | age of client $k$'s local model |
| $b_l \in \mathbb{Z}$ | lower limit on the age difference between $a$ and $a_k$. |
| $b_u \in \mathbb{Z}, b_u > b_l$ | upper limit on the age difference between $a$ and $a_k$. |

## A.2 Variants of FedAvg

Fig. A.1 shows the same plots as Fig. 5.2 but these are smoothed. The smoothing function is equivalent to how tensorboard does smoothing[1], with a weighting factor of 0.2.

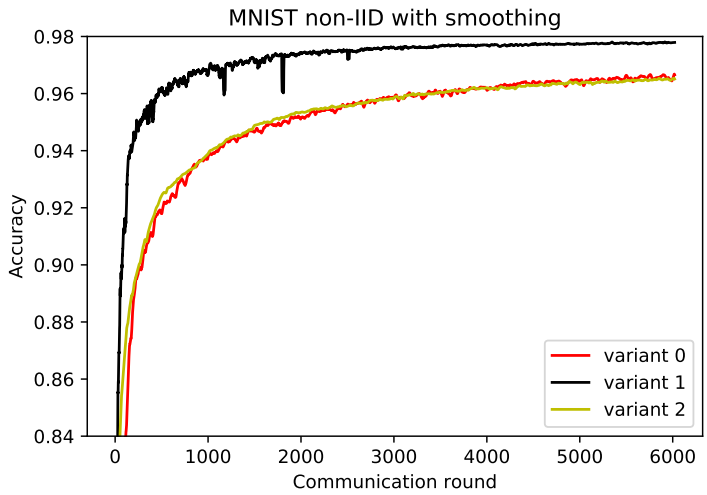



**Figure A.1:** Three different variants of FedAvg. All of them uses $C = 0.1$, $E = 5$, $B = 20$, $\eta = 0.05$, $\lambda = 10^{-5}$.

## A.3 Hyperparameter search for FedAvg

Table A.1, A.2, and A.3 show all runs made for the hyperparameter search for FedAvg. A more detailed explanation of the columns can be found in Sec. 6.1.

**Table A.1:** Exponential parameter search for learning rate $\eta$ for FedAvg.

| $\eta$ | max(acc) | min(err) | avg(acc) | sum(drop) |
|---|---|---|---|---|
| 0.00015 | 0.9052 | 0.3448 | 0.8345 | 4.216 |
| 0.00046 | 0.9087 | 0.3268 | 0.8336 | 14.89 |
| 0.0014 | 0.9192 | 0.2673 | 0.8413 | 19.24 |
| 0.0041 | 0.9419 | 0.192 | 0.8766 | 14.51 |
| 0.012 | 0.9586 | 0.1367 | 0.9146 | 9.19 |
| 0.037 | 0.9727 | 0.1107 | 0.9429 | 5.429 |

[1]Tensorboard's smoothing function: github.com/tensorflow/tensorflow

**Table A.2:** Linear parameter search for learning rate $\eta$ for FedAvg.

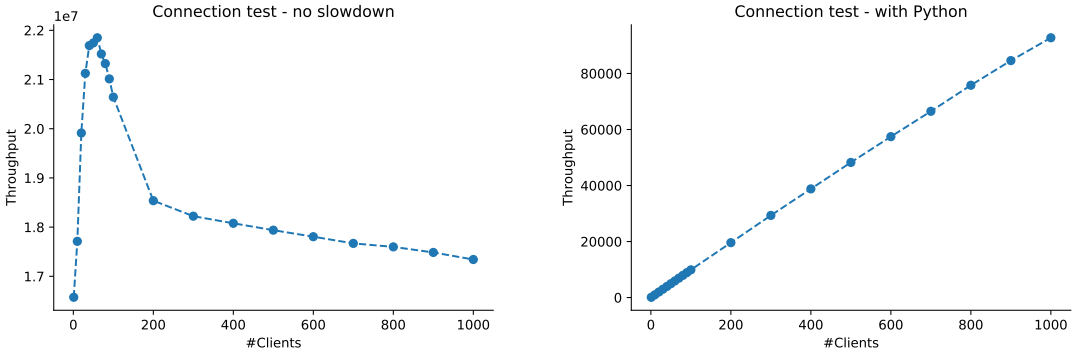
| $\eta$ | max(acc) | min(err) | avg(acc) | sum(drop) |
|---|---|---|---|---|
| 0.02 | 0.9646 | 0.1264 | 0.929 | 7.044 |
| 0.03 | 0.9699 | 0.1148 | 0.9391 | 5.638 |
| 0.04 | 0.973 | 0.1091 | 0.9452 | 4.833 |
| 0.05 | 0.9738 | 0.1083 | 0.9481 | 4.936 |
| 0.06 | 0.9761 | 0.1053 | 0.9513 | 4.489 |
| 0.07 | 0.9759 | 0.1049 | 0.9518 | 4.908 |
| 0.08 | 0.9755 | 0.1453 | 0.9455 | 6.482 |
| 0.09 | 0.9307 | 0.2296 | 0.2515 | 6.804 |

**Table A.3:** Random search for $\eta$ and $\lambda$ for FedAvg.

| $\eta$ | $\lambda$ | max(acc) | min(err) | avg(acc) | sum(drop) |
|---|---|---|---|---|---|
| 0.037 | 4.9e-06 | 0.9742 | 0.1078 | 0.9525 | 5.973 |
| 0.047 | 2.5e-05 | 0.9747 | 0.09802 | 0.9567 | 5.054 |
| 0.05 | 1.6e-06 | 0.9767 | 0.101 | 0.9582 | 5.014 |
| 0.056 | 1.3e-08 | 0.9773 | 0.1033 | 0.9585 | 5.847 |
| 0.061 | 1.8e-06 | 0.9777 | 0.09984 | 0.9607 | 4.749 |
| 0.068 | 2.8e-07 | 0.978 | 0.109 | 0.96 | 5.058 |
| 0.068 | 8.0e-07 | 0.9778 | 0.1026 | 0.961 | 4.769 |
| 0.081 | 3.2e-06 | 0.9784 | 0.09968 | 0.9629 | 3.986 |
| 0.082 | 2.1e-08 | 0.9787 | 0.1043 | 0.9618 | 4.791 |
| 0.084 | 1.0e-04 | 0.9761 | 0.09407 | 0.9602 | 4.717 |
| 0.088 | 3.2e-06 | 0.9788 | 0.09794 | 0.9638 | 4.115 |
| 0.091 | 4.1e-05 | 0.9775 | 0.1004 | 0.9623 | 4.728 |
| 0.094 | 4.5e-04 | 0.9732 | 0.09498 | 0.9574 | 4.959 |
| 0.098 | 2.4e-08 | 0.979 | 0.1325 | 0.9562 | 7.108 |
| 0.11 | 2.2e-07 | 0.9792 | 0.09315 | 0.9658 | 3.444 |
| 0.12 | 2.3e-08 | 0.7447 | 0.6367 | 0.1039 | 1.319 |
| 0.12 | 1.2e-07 | 0.979 | 0.1031 | 0.964 | 4.493 |
| 0.13 | 3.5e-04 | 0.9748 | 0.09387 | 0.9602 | 4.68 |
| 0.14 | 7.8e-04 | 0.6962 | 0.8643 | 0.09896 | 0.8438 |
| 0.14 | 7.9e-05 | 0.8854 | 0.3951 | 0.1047 | 2.877 |

## A.4 Framework scalability experiments

Fig. A.2 shows the results of the two experiments discussed in Sec. 4.8. Each experiment used 20 different client configurations, ranging from a single to 1000 clients. Four benchmarks were performed per client configuration and each benchmark lasted for five minutes. The four iterations are averaged to produce the measured values in Fig.A.2.




**(a)** **(b)**

**Figure A.2:** Connection tests simulated on one machine with up to 1000 clients connected simultaneously to the Erlang server node. Simulated locally on one machine. (a) The simulation computer is quickly overwhelmed when latency is low. (b) Throughput scales linearly with the number of clients when additional delay is introduced.

Both experiments measure throughput as the number of distributed operations. In our case, a distributed operation is counted whenever the server receives a pong response from a client. Specifically, a benchmark was performed as follows. The user tells the server to run a benchmark for 5 minutes with a throughput counter initially set to zero. The server records the start time and sends out ping messages to all connected clients, collecting responses as they arrive. Clients then immediately send a pong response to the server upon receiving a ping message. Whenever the server receives a pong response, it asserts that the benchmark is still active and then increments the throughput counter by one. Once the benchmark has ended, the throughput counter is not incremented but instead reported to the user. Given that all clients have responded, and the benchmark has not ended, the server sends out new ping messages.

## A.5 Statements regarding the CO-OP age filter

This section is a collection of remarks and one proposition about CO-OP's age filter.

**Remark 1.** *The intuition behind $b_l$ is: a client cannot do another normal update until $b_l$ other clients have done one normal update each.*
**Remark 2.** *The intuition behind $b_u$ is: a client that just did a normal update will be outdated if it does not make another update before $b_u$ normal updates have been done by other clients.*
**Proposition 1.** *It is possible for the CO-OP algorithm to deadlock if $b_u < 2b_l$, and $K \geq b_u - b_l + 1$.*

*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:

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

Assuming (A.1) 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 $a = b_u + 1$, then they are considered to be outdated and set their $a_k = a$. However, we are now in a state where all clients are considered to be overactive. In other words, it is possible for CO-OP to deadlock if $b_u < 2b_l$. □

**Remark 3.** *The constraint $b_u - b_l \geq K - 1$ is a special case of the constraint $b_u \geq 2b_l$ where $b_l = K - 1$.*