Gossip Learning as a Decentralized Alternative to Federated Learning[⋆]

István Hegedűs[1] [0000−0002−5356−2192], Gábor Danner[1] [0000−0002−9983−1060], and Márk Jelasity[1,2] [0000−0001−9363−1482]

[1] University of Szeged, Szeged, Hungary
[2] MTA SZTE Research Group on Artificial Intelligence, Szeged, Hungary

Abstract. Federated learning is a distributed machine learning approach for computing models over data collected by edge devices. Most importantly, the data itself is not collected centrally, but a master-worker architecture is applied where a master node performs aggregation and the edge devices are the workers, not unlike the parameter server approach. Gossip learning also assumes that the data remains at the edge devices, but it requires no aggregation server or any central component. In this empirical study, we present a thorough comparison of the two approaches. We examine the aggregated cost of machine learning in both cases, considering also a compression technique applicable in both approaches. We apply a real churn trace as well collected over mobile phones, and we also experiment with different distributions of the training data over the devices. Surprisingly, gossip learning actually outperforms federated learning in all the scenarios where the training data are distributed uniformly over the nodes, and it performs comparably to federated learning overall.

1 Introduction

Performing data mining over data collected by edge devices, most importantly, mobile phones, is of very high interest [17]. Collecting such data at a central location has become more and more problematic in the past years due to novel data protection rules [9] and in general due to the increasing public awareness to issues related to data handling. For this reason, there is an increasing interest in methods that leave the raw data on the device and process it using distributed aggregation.

Google introduced *federated learning* to answer this challenge [12, 13]. This approach is very similar to the well-known parameter server architecture for distributed learning [7] where worker nodes store the raw data. The parameter

⋆ The original publication is available at link.springer.com. In Proc. DAIS 2019, LNCS 11534, pp. 74–90, 2019. DOI: 10.1007/978-3-030-22496-7_5. This work was supported by the Hungarian Government and the European Regional Development Fund under the grant number GINOP-2.3.2-15-2016-00037 (“Internet of Living Things”) and by the Hungarian Ministry of Human Capacities (grant 20391-3/2018/FEKUSTRAT).

server maintains the current model and regularly distributes it to the workers who in turn calculate a gradient update and send it back to the server. The server then applies all the updates to the central model. This is repeated until the model converges. In federated learning, this framework is optimized so as to minimize communication between the server and the workers. For this reason, the local update calculation is more thorough, and compression techniques can be applied when uploading the updates to the server.

In addition to federated learning, *gossip learning* has also been proposed to address the same challenge [10, 15]. This approach is fully decentralized, no parameter server is necessary. Nodes exchange and aggregate models directly. The advantages of gossip learning are obvious: since no infrastructure is required, and there is no single point of failure, gossip learning enjoys a significantly *cheaper scalability and better robustness*. The key question, however, is how the two approaches compare in terms of performance. This is the question we address in this work. To be more precise, we compare the two approaches in terms of convergence time and model quality, assuming that both approaches utilize the same amount of communication resources in the same scenarios.

To make the comparison as fair as possible, we make sure that the two approaches differ mainly in their communication patterns. However, the computation of the local update is identical in both approaches. Also, we apply subsampling to reduce communication in both approaches, as introduced in [12] for federated learning. Here, we adapt the same technique for gossip learning.

We learn linear models using stochastic gradient descent (SGD) based on the logistic regression loss function. For realistic simulations, we apply smartphone churn traces collected by the application Stunner [2]. We note that both approaches offer mechanisms for explicit privacy protection, apart from the basic feature of not collecting data. In federated learning, Bonawitz et al. [3] describe a secure aggregation protocol, whereas for gossip learning one can apply the methods described in [4]. Here, we are concerned only with the efficiency of the different communication patterns and do not compare security mechanisms.

The result of our comparison is that gossip learning is in general comparable to the centrally coordinated federated learning approach, and in many scenarios gossip learning actually outperforms federated learning. This result is rather counter-intuitive and suggests that decentralized algorithms should be treated as first class citizens in the area of distributed machine learning overall, considering the additional advantages of decentralization.

The outline of the paper is as follows. Section 2 describes the basics of federated learning and gossip learning. Section 3 describes the specific algorithmic details that were applied in our comparative study, in particular, the management of the learning rate parameter and the subsampling compression techniques. Section 4 presents our results.

2 Background

Classification is a fundamental problem in machine learning. Here, a data set $D = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ of n examples is given, where an example is represented by a feature vector $x \in R^d$ and the corresponding class label $y \in C$, where d is the dimension of the problem and C is the set of class labels. The problem of classification is often expressed as finding the parameters w of a function $f_w : R^d \to C$ that can correctly classify as many examples as possible in D, as well as outside D (this latter property is called generalization). Expressed formally, the objective function $J(w)$ captures the error of the model parameters w, and we wish to minimize $J(w)$ in w:

$$w^* = \arg\min_w J(w) = \arg\min_w \frac{1}{n}\sum_{i=1}^{n} \ell(f_w(x_i), y_i) + \frac{\lambda}{2}\|w\|^2, \tag{1}$$

where $\ell()$ is the loss function (the error of the prediction), $\|w\|^2$ is the regularization term, and λ is the regularization coefficient. By keeping the model parameters small, regularization helps in avoiding overfitting to the training set.

Perhaps the simplest algorithm to approximate w^* is the gradient descent method. Here, we start with a random weight vector w_0. In each iteration, we compute w_{t+1} based on w_t by finding the gradient of the objective function at w_t and making a step towards the direction opposite to the gradient. One such iteration is called a gradient update. Formally,

$$w_{t+1} = w_t - \eta_t(\frac{\partial J}{\partial w}(w_t)) = w_t - \eta_t(\lambda w_t + \frac{1}{n}\sum_{i=1}^{n}\frac{\partial \ell(f_w(x_i), y_i)}{\partial w}(w_t)), \tag{2}$$

where η_t is the learning rate at iteration t. Stochastic gradient descent (SGD) is similar, only we use a single example (x_i, y_i) instead of the entire database to perform an update:

$$w_{t+1} = w_t - \eta_t(\lambda w_t + \frac{\partial \ell(f_w(x_i), y_i)}{\partial w}(w_t)). \tag{3}$$

It is also usual to apply a so called *minibatch update*, in which more than one example is used, but not the entire database.

In this study we use *logistic regression* as our machine learning model, where the specific form of the objective function is given by

$$J(w) = -\frac{1}{n}\sum_{i=1}^{n} \ln P(y_i|x_i, w) + \frac{\lambda}{2}\|w\|^2, \tag{4}$$

where $y_i \in \{0, 1\}$, $P(0|x_i, w) = (1+\exp(w^T x))^{-1}$ and $P(1|x_i, w) = 1-P(0|x_i, w)$.

2.1 Federated Learning

The pseudocode of the federated learning algorithm [12, 13] is shown in Algorithm 1 (master) and Algorithm 2 (worker). The master periodically sends

Algorithm 1 Federated Learning Master

```
 1: $(t, w) \leftarrow$ init()
 2: loop
 3:     for every node $i$ in parallel do          ▷ non-blocking (in separate thread(s))
 4:         send $(t, w)$ to $i$
 5:         receive $(n_i, h_i)$ from $i$          ▷ $n_i$: example count at $i$; $h_i$: model gradient
 6:     end for
 7:     wait($\mathit{\Delta}_f$)                   ▷ the round length
 8:     $n \leftarrow \frac{1}{|\mathcal{I}|}\sum_{i\in\mathcal{I}} n_i$          ▷ $\mathcal{I}$: nodes that returned a model in this round
 9:     $t \leftarrow t + n$
10:     $h \leftarrow$aggregate($\{h_i : i \in \mathcal{I}\}$)
11:     $w \leftarrow w + h$
12: end loop
```

Algorithm 2 Federated Learning Worker

```
1: procedure onReceiveModel($t, w$)
2:     $(t', w') \leftarrow$update($(t, w), D_k$)          ▷ $D_k$: the local database of examples
3:     $(n, h) \leftarrow (t' - t, w' - w)$                ▷ $n$: the number of local examples
4:     send ($n$, compress($h$)) to master
5: end procedure
```

the current model w to all the workers asynchronously in parallel and collects the answers from the workers. Any answers from workers arriving with a delay larger than $\mathit{\Delta}_f$ are simply discarded. After $\mathit{\Delta}_f$ time units have elapsed, the master aggregates the received gradients and updates the model. We also send and maintain the model age t (based on the average number of examples used for training) in a similar fashion, to enable the use of dynamic learning rates in the local learning. These algorithms are very generic, the key characteristics of federated learning lie in the details of the update method (line 2 of Algorithm 2) and the compression mechanism (line 4 of Algorithm 2 and line 10 of Algorithm 1). The update method is typically implemented through a minibatch gradient descent algorithm that operates on the local data, initialized with the received model w. The details of our implementation of the update method and compression is presented in Section 3.

Algorithm 3 Gossip Learning Framework

```
 1: $(t_k, w_k) \leftarrow$ init()
 2: loop
 3:     wait($\mathit{\Delta}_g$)
 4:     $p \leftarrow$ select()
 5:     send ($t_k$, compress($w_k$)) to $p$
 6: end loop
 7: procedure onReceiveModel($t_r, w_r$)
 8:     $(t_k, w_k) \leftarrow$merge($(t_k, w_k), (t_r, w_r)$)
 9:     $(t_k, w_k) \leftarrow$update($(t_k, w_k), D_k$)
10: end procedure
```

Algorithm 4 Model update rule

```
1: procedure UPDATE(($t, w$), $D$)
2:     for all batch $B \subseteq D$ do                         ▷ D is split into batches
3:         $t \leftarrow t + |B|$
4:         $w \leftarrow w - \eta_t \sum_{(x,y)\in B}(\frac{\partial \ell(f_w(x),y)}{\partial w}(w) + \lambda w)$
5:     end for
6:     return ($t, w$)
7: end procedure
```

Algorithm 5 Model initialization

```
1: procedure INIT()
2:     $t \leftarrow 0$
3:     $w \leftarrow \mathbf{0}$                        ▷ $\mathbf{0}$ denotes the vector of all zeros
4:     return ($t, w$)
5: end procedure
```

2.2 Gossip Learning

Gossip Learning is a method for learning models from fully distributed data without central control. Each node k runs Algorithm 3. First, the node initializes a local model w_k (and its age t_k). This is then periodically sent to another node in the network. (Note that these cycles are not synchronized.) The node selection is supported by a so-called sampling service [11,16]. Upon receiving a model w_r, the node merges it with the local model, and updates it using the local data set D_k. Merging is typically achieved by averaging the model parameters; see Section 3 for specific implementations. In the simplest case, the received model merely overwrites the local model. This mechanism results in the models taking random walks in the network and being updated when visiting a node. The possible update methods are the same as in the case of federated learning, and compression can be applied as well.

3 Algorithms

In this section we describe the details of the UPDATE, INIT, COMPRESS, AGGREGATE, and MERGE methods. Methods UPDATE, INIT and COMPRESS are shared among federated learning and gossip learning. In all the cases we used the implementations in Algorithms 4 and 5. In the minibatch update we compute the sum instead of the average to give an equal weight to all the examples irrespective of batch size. (Note that even if the minibatch size is fixed, actual sizes will vary because the number of examples at a given node is normally not divisible with the nominal batch size.) We used the dynamic learning rate $\eta_t = \eta/t$, where t is the number of instances the model was trained on.

Method AGGREGATE is used in Algorithm 1. Its function is to decompress and aggregate the received gradients encoded with COMPRESS. When there is no actual compression (COMPRESSNONE in Algorithm 7), simply the average of

Algorithm 6 Various versions of the aggregate function

```
procedure AGGREGATEDEFAULT($H$)                          ▷ Average of gradients
    return $\frac{1}{|H|}\sum_{h \in H} h$
end procedure

procedure AGGREGATESUBSAMPLED($H$)                       ▷ Restore expected value
    return $\frac{d}{s|H|}\sum_{h \in H} h$    ▷ s: number of model parameters kept by subsampling
end procedure

procedure AGGREGATESUBSAMPLEDIMPROVED($H$)
    $h' \leftarrow \mathbf{0}$
    for $i \in \{1, ..., d\}$ do
        $H_i \leftarrow \{h : h \in H \wedge h[i] \neq 0\}$ ▷ $h[i]$ refers to the $i$th element of the vector $h$
        $h'[i] \leftarrow \frac{1}{|H_i|}\sum_{h \in H} h[i]$                 ▷ skipped if $|H_i| = 0$
    end for
    return $h'$
end procedure
```

Algorithm 7 Various versions of the compress function

```
procedure COMPRESSNONE($h$)
    return $h$
end procedure

procedure COMPRESSSUBSAMPLING($h$)
    $h' \leftarrow \mathbf{0}$
    $X \leftarrow$ random subset of $\{1, ..., d\}$ of size $s$
    for $i \in X$ do
        $h'[i] \leftarrow h[i]$
    end for
    return $h'$
end procedure
```

gradients is taken (AGGREGATEDEFAULT in Algorithm 6). The compression technique we employed is subsampling [13]. When using subsampling, workers do not send all of the model parameters back to the master, but only random subsets of a given size (see COMPRESSSUBSAMPLING). Note that the indices need not be sent, instead, we can send the random seed used to select them. The missing values are treated as zero. Due to this, the gradient average needs to be scaled as shown in AGGREGATESUBSAMPLED to create an unbiased estimator of the original gradient. We introduce a slight improvement to this scaling method in AGGREGATESUBSAMPLEDIMPROVED. Here, instead of scaling based on the theoretical probability of including a parameter, we calculate the actual average for each parameter separately based on the number of the gradients that contain the given parameter.

In gossip learning, MERGE is used to combine the local model with the incoming one. In the simplest variation, the local model is discarded in favor of

Algorithm 8 Various versions of the merge function

```
procedure mergeNone($(t,w),(t_r,w_r)$)
    return $(t_r,w_r)$
end procedure

procedure mergeAverage($(t,w),(t_r,w_r)$)
    $a \leftarrow \frac{t_r}{t+t_r}$
    $t \leftarrow max(t,t_r)$
    $w \leftarrow (1-a)w + aw_r$
    return $(t,w)$
end procedure

procedure mergeSubsampled($(t,w),(t_r,w_r)$)    ▷ averages non-zero values only
    $a \leftarrow \frac{t_r}{t+t_r}$
    $t \leftarrow max(t,t_r)$
    for $i \in \{1,...,d\}$ do
        if $w_r[i] \neq 0$ then    ▷ $w[i]$ refers to the $i$th element of the vector $w$
            $w[i] \leftarrow (1-a)w[i] + aw_r[i]$
        end if
    end for
    return $(t,w)$
end procedure
```

the received model (see mergeNone in Algorithm 8). It is usually a better idea to take the average of the parameter vectors [15]. We use average weighted by model age (see mergeAverage). Subsampling can be used with gossip learning as well, in which case mergeSubsampled must be used, which considers only the received parameters.

4 Experiments

4.1 Datasets

We used three datasets from the UCI machine learning repository [8] to test the performance of our algorithms. The first is the Spambase (SPAM E-mail Database) dataset containing a collection of emails. Here, the task is to decide whether an email is spam or not. The emails are represented by high level features, mostly word or character frequencies. The second dataset is Pendigits (Pen-Based Recognition of Handwritten Digits) that contains downsampled images of 4×4 pixels of digits from 0 to 9. The third is the HAR (Human Activity Recognition Using Smartphones) [1] dataset, where human activities (walking, walking upstairs, walking downstairs, sitting, standing and laying) were monitored by smartphone sensors (accelerometer, gyroscope and angular velocity). High level features were extracted from these measurement series.

The main properties, such as size or number of features, are presented in Table 1. In our experiments we standardized the feature values, that is, shifted

Table 1. Data set properties

	Spambase	Pendigits	HAR
Training set size	4140	7494	7352
Test set size	461	3498	2947
Number of features	57	16	561
Number of classes	2	10	6
Class-label distribution	$\approx$ 6:4	$\approx$ uniform	$\approx$ uniform
Parameter η	1E+4	1E+4	1E+2
Parameter λ	1E-6	1E-4	1E-2

and scaled them to have a mean of 0 and a variance of 1. Note that the standardization can be approximated by the nodes in the network locally if the approximation of the statistics of the features are fixed and known, which can be ensured in a fixed application.

In our simulation experiments, each example in the training data was assigned to one node when the number of nodes was 100. This means that, for example, with the HAR dataset each node gets 73.5 examples on average. When the network size is 1000, we replicate the examples, that is, each example is assigned to 10 different nodes. As for the distribution of class labels on the nodes, we applied two different setups. The first one is *uniform assignment*, which means that we assigned the examples to nodes at random independently of class label. The number of samples assigned to each node was the same (to be more precise, it differed by at most one due to the number of samples not being divisible by 100).

The second one is *single class assignment* when every node has examples only from a single class. Here, the different class labels are assigned uniformly to the nodes, and then the examples with a given label are assigned to one of the nodes with the same label, uniformly. These two assignment strategies represent the two extremes in any real application. In a realistic setting the class labels will likely be biased but much less so than in the case of the single class assignment scenario.

4.2 System model

In our simulation experiments, we used a fixed random k-out overlay network, with $k = 20$. That is, every node had $k = 20$ fixed random neighbors. Simulations were performed with a network size of 100 and 1000 nodes. In the churn-free scenario, every node stayed online for the whole experiment. The churn scenario is based on a real trace gathered from smartphones (see Section 4.3 below). We assumed that a message is successfully delivered if and only if both the sender and the receiver remains online during the transfer. We also assume that the nodes are able to detect which of their neighbors are online at any given time with a delay that is negligible compared to the transfer time of a model.

We assumed uniform upload and download bandwidths for the nodes, and infinite bandwidth on the side of the server. Note that the latter assumption favors federated learning, as gossip learning does not use a server. The uniform bandwidth assumption is motivated by the fact that it is likely that in a real application there will be a configured (uniform) bandwidth cap that is significantly lower than the average available bandwidth. The transfer time of a full model was assumed to be 172 seconds (irrespective of the dataset used) in the *long transfer time* scenario, and 17.2 seconds in the *short transfer time* scenario. This allowed for around 1,000 and 10,000 iterations over the course of 48 hours, respectively.

The cycle length parameters Δ_g and Δ_f were set based on the constraint that in the two algorithms the nodes should be able to exploit all the available bandwidth. In our setup this also means that the two algorithms transfer the same number of bits overall in the network in the same time-window. This will allow us to make fair comparisons regarding convergence dynamics. The gossip cycle length Δ_g is thus exactly the transfer time of a full model, that is, nodes are assumed to send messages continuously. The cycle length Δ_f of federated learning is the round-trip time, that is, the sum of the upstream and downstream transfer times. When compression is used, the transfer time is proportionally less as defined by the compression rate. Note, however, that in federated learning the master always sends the full model to the workers, only the upstream transfer is compressed.

It has to be noted that we assume much longer transfer times than what would be appropriate for the actual models in our simulation. To put it differently, in our simulations we pretend that our models are very large. This is because in the churn scenario if the transfer times are very short, the network hardly changes during the learning process, so effectively we learn over a static subset of the nodes. Long transfer times, however, make the problem more challenging because many transfers will fail, just like in the case of very large machine learning models such as deep neural networks. In the case of the no-churn scenario this issue is completely irrelevant, since the dynamics of convergence are identical apart from scaling time.

4.3 Smartphone traces

The trace we used was collected by a locally developed openly available smartphone app called STUNner, as described previously [2]. In a nutshell, the app monitors and collects information about charging status, battery level, bandwidth, and NAT type.

We have traces of varying lengths taken from 1191 different users. We divided these traces into 2-day segments (with a one-day overlap), resulting in 40,658 segments altogether. With the help of these segments, we were able to simulate a virtual 48-hour period by assigning a different segment to each simulated node.

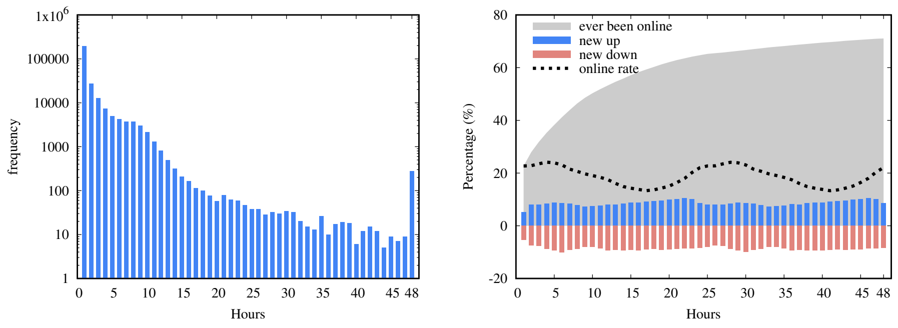

Fig. 1. Online session length distribution (left) and dynamic trace properties (right)

To ensure our algorithm is phone and user friendly, we defined a device to be online (available) when it has been on a charger and connected to the internet for at least a minute, hence we never use battery power at all. In addition, we also treated those users as offline who had a bandwidth of less than 1 Mbit/s.

Figure 1 illustrates some of the properties of the trace. The plot on the right illustrates churn via showing, for every hour, what percentage of the nodes left, or joined the network (at least once), respectively. We can also see that at any given moment about 20% of the nodes are online. The average session length is 81.368 minutes.

4.4 Hyperparameters and Algorithms

The learning rate η and regularization coefficient λ were optimized using grid search assuming the no-failure scenario, no compression, and uniform assignment. The resulting values are shown in Table 1. These hyperparameters depend only on the database, they are robust to the selection of the algorithm. Minibatches of size 10 were used in each scenario. We used logistic regression as our learning algorithm, embedded in a one-vs-all meta-classifier.

4.5 Results

We ran the simulations using PeerSim [14]. We measure learning performance with the help of the 0-1 loss, which gives the proportion of the misclassified examples in the test set. In the case of gossip learning the loss is defined as the average loss over the online nodes.

First, we compare the two aggregation algorithms for subsampled models in Algorithm 6 (Figure 2) in the no-failure scenario. The results indicate a slight advantage of AGGREGATESUBSAMPLINGIMPROVED, although the performance depends on the database. In the following we will apply AGGREGATESUBSAMPLINGIMPROVED as our implementation of method AGGREGATE.

The comparison of the different algorithms and subsampling probabilities is shown in Figure 3. The stochastic gradient descent (SGD) method is also shown,

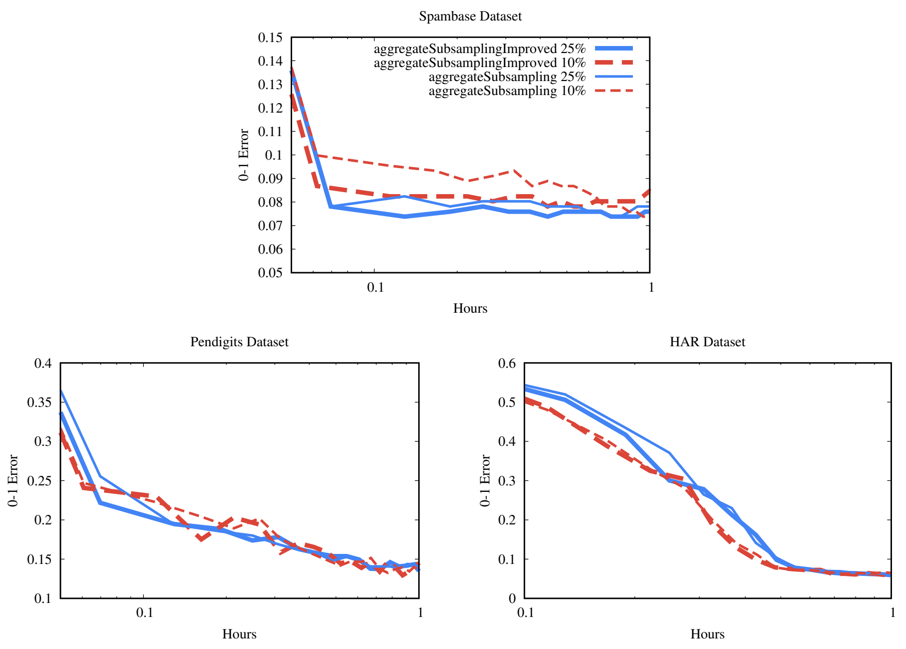

Fig. 2. Federated learning, 100 nodes, long transfer time, no failures, different aggregation algorithms and subsampling probabilities.

which was implemented by gossip learning with no merging (using mergeNone). Clearly, the parallel methods are all better than SGD. Also, it is very clear that subsampling helps both federated learning and gossip learning. However, gossip learning benefits much more from it. The reason is that in the case of federated learning subsampling is implemented only in the worker master direction, the master sends the full model back to the workers [12]. However, in gossip learning, subsampling can be applied to all the messages.

Most importantly, gossip learning clearly *outperforms* federated learning in the case of high compression rates (low sampling probability) over two of the three datasets, and it is competitive on the remaining dataset as well. This was not expected, as gossip learning is fully decentralized, so the aggregation is clearly delayed compared to federated learning. Indeed, with no compression, federated learning performs better. However, with high compression rates, slower aggregation is compensated by a higher communication efficiency. Figure 3 also illustrates scaling. As we can see, the performance with 100 and 1000 nodes is practically identical for both algorithms.

Figure 4 contains our results with the churn trace. In the first hour, the two algorithms behave just like in the no-churn scenario. On the longer range, clearly, federated learning tolerates the churn better. This is because in federated learning nodes always work with the freshest possible models that they receive

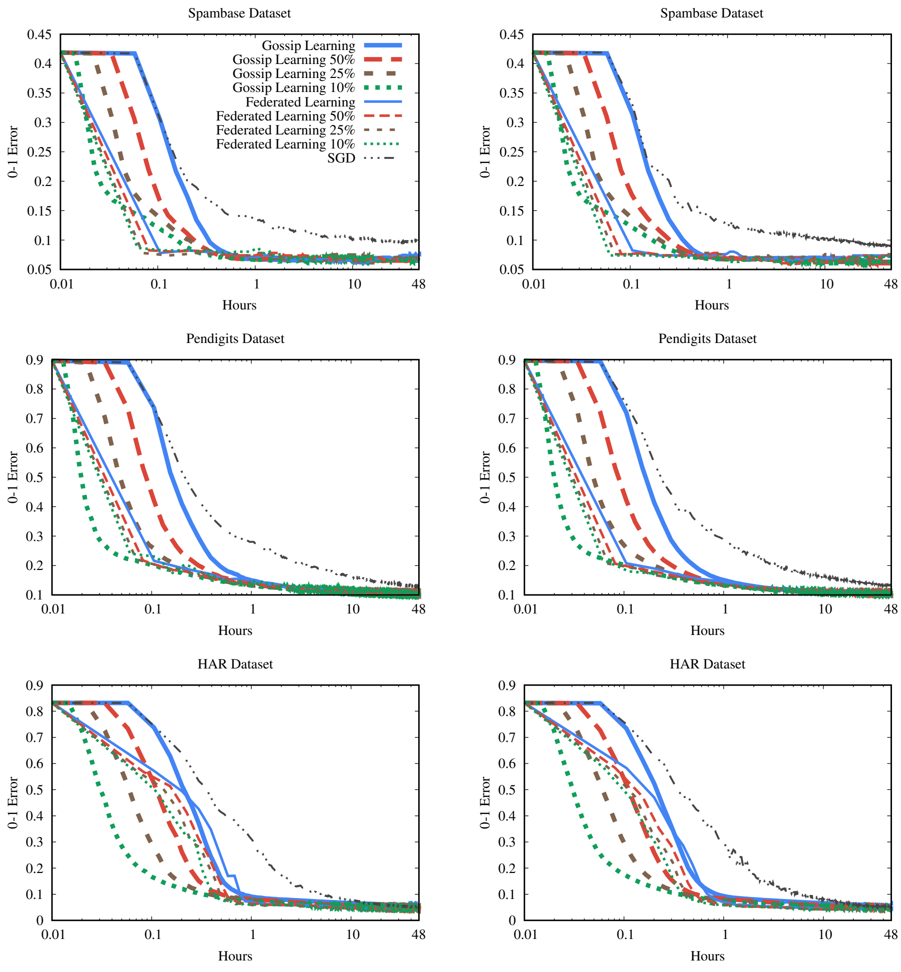

Fig. 3. Federated learning and gossip learning with 100 (left) and 1000 (right) clients, long transfer time, no failures, with different subsampling probabilities. Minibatch Stochastic Gradient Descent (SGD) is implemented by gossip learning with no merging (using MERGENONE).

from the master, even right after coming back online. In gossip learning, outdated models could temporarily participate in the optimization, albeit with a smaller weight. In this study we did not invest any effort into mitigating this effect, but outdated models could potentially be removed with more aggressive methods as well.

We also include an artificial trace scenario, where online session lengths are exponentially distributed following the same expected length (81 minutes) as in the smartphone trace. The offline session length is set so we have 10% of

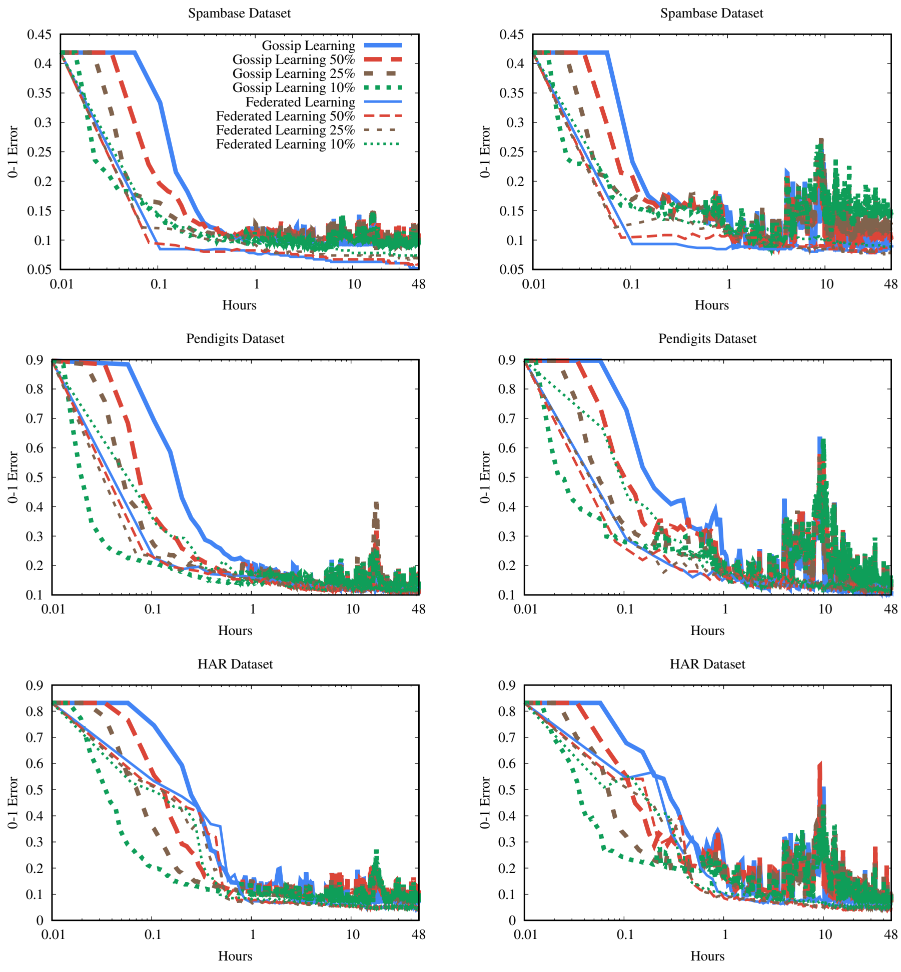

Fig. 4. Federated learning and gossip learning over the smartphone trace (left) and an artificial exponential trace (right), long transfer time, with different subsampling probabilities.

the nodes spending any given federated learning round online in expectation, assuming no compression. This is to reproduce similar experiments in [13]. The results are similar to those over the smartphone trace, only the noise is larger for gossip learning, because the exponential model results in an unrealistically large variance in session lengths.

Figure 5 shows the convergence dynamics when we assume short transfer times (see Section 4.2). Clearly, the scenarios without churn result in the same dynamics (apart from a scaling factor) as the scenarios with long transfer time.

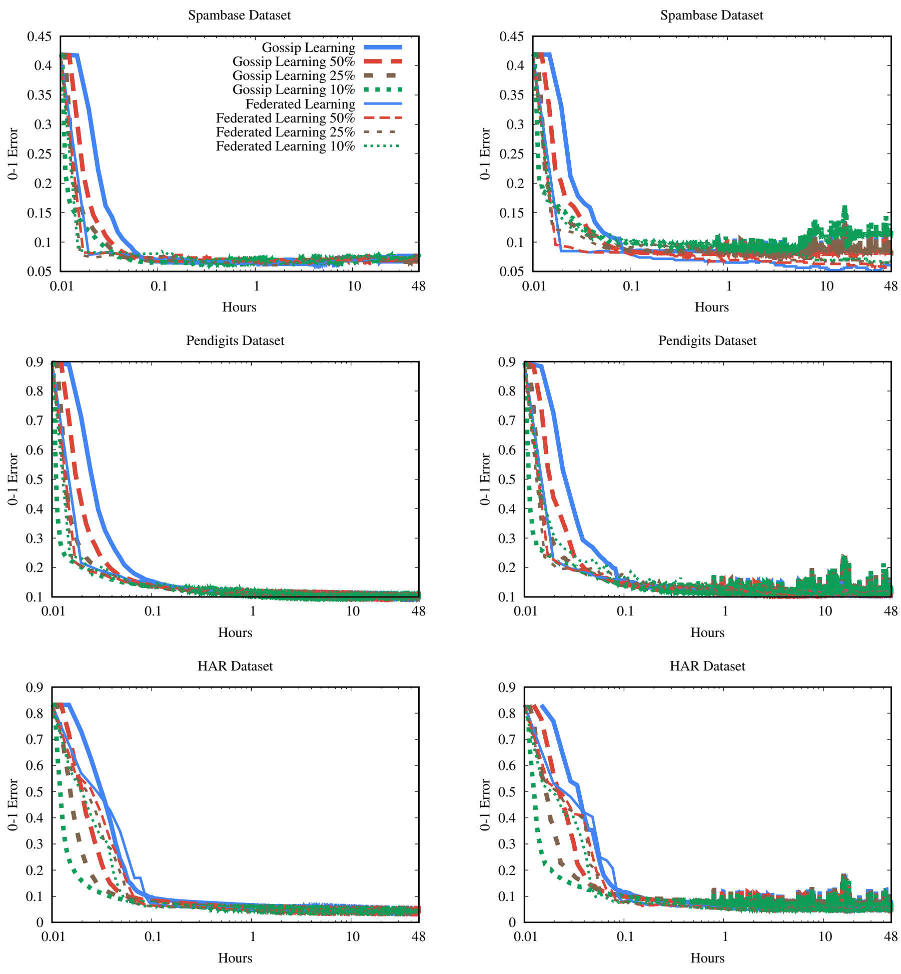

Fig. 5. Federated learning and gossip learning with no churn (left) and over the smartphone trace (right), short transfer time, with different subsampling probabilities.

The algorithms are somewhat more robust to churn in this case, since the nodes are more stable relative to message transfer time.

Figure 6 contains the results of our experiments with the single class assignment scenario, as described in Section 4.1. In this extreme scenario, the learning problem becomes much harder. Still, gossip learning remains competitive in the case of high compression rates.

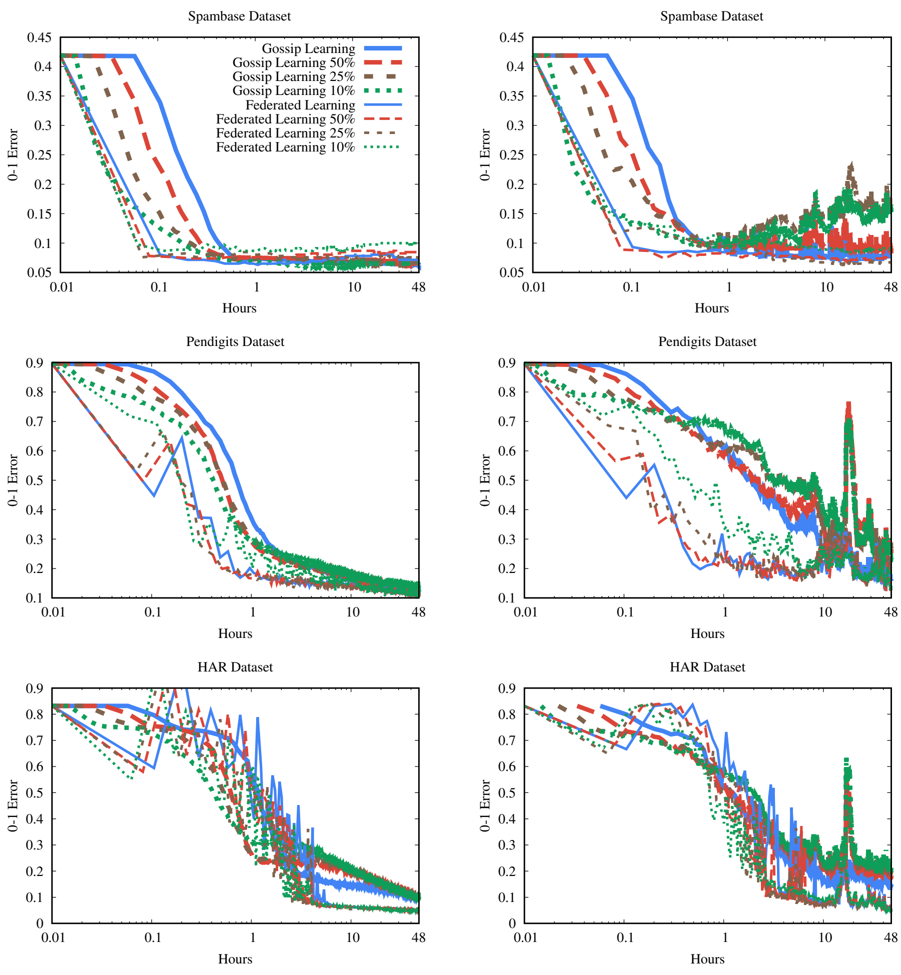

Fig. 6. Federated learning and gossip learning with no churn (left) and over the smartphone trace (right), long transfer time, single class assignment, with different subsampling probabilities.

5 Conclusions

Here, our goal was to compare federated learning and gossip learning in terms of efficiency. We designed an experimental study to answer this question. We compared the convergence speed of the two approaches under the assumption that both methods use the available bandwidth, resulting in an identical overall bandwidth consumption.

We found that in the case of uniform assignment, gossip learning is not only comparable to the centralized federated learning, but it even outperforms it under the highest compression rate settings. In every scenario we examined, gossip learning is comparable to federated learning. We add that this result relies on our experimental assumptions. For example, if one considers the download traffic to be essentially free in terms of bandwidth and time then federated learning is more favorable. This, however, is not a correct approach because it hides the costs at the side of the master node. For this reason, we opted for modeling the download bandwidth to be identical to the upload bandwidth, but still assuming an infinite bandwidth at the master node.

As for future work, the most promising direction is the design and evaluation of more sophisticated compression techniques [5] for both federated and gossip learning. Also, in both cases, there is a lot of opportunity to optimize the communication pattern by introducing asynchrony to federated learning, or adding flow control to gossip learning [6].

References

1. Anguita, D., Ghio, A., Oneto, L., Parra, X., Reyes-Ortiz, J.L.: A public domain dataset for human activity recognition using smartphones. In: 21th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN) (2013)
2. Berta, Á., Bilicki, V., Jelasity, M.: Defining and understanding smartphone churn over the internet: a measurement study. In: Proceedings of the 14th IEEE International Conference on Peer-to-Peer Computing (P2P 2014). IEEE (2014)
3. Bonawitz, K., Ivanov, V., Kreuter, B., Marcedone, A., McMahan, H.B., Patel, S., Ramage, D., Segal, A., Seth, K.: Practical secure aggregation for federated learning on user-held data. In: NIPS Workshop on Private Multi-Party Machine Learning (2016)
4. Danner, G., Berta, Á., Hegedűs, I., Jelasity, M.: Robust fully distributed minibatch gradient descent with privacy preservation. Security and Communication Networks 2018, 6728020 (2018)
5. Danner, G., Jelasity, M.: Robust decentralized mean estimation with limited communication. In: Aldinucci, M., Padovani, L., Torquati, M. (eds.) Euro-Par 2018. Lecture Notes in Computer Science, vol. 11014, pp. 447–461. Springer International Publishing (2018)
6. Danner, G., Jelasity, M.: Token account algorithms: The best of the proactive and reactive worlds. In: Proceedings of The 38th International Conference on Distributed Computing Systems (ICDCS 2018). pp. 885–895. IEEE Computer Society (2018)
7. Dean, J., Corrado, G.S., Monga, R., Chen, K., Devin, M., Le, Q.V., Mao, M.Z., Ranzato, M., Senior, A., Tucker, P., Yang, K., Ng, A.Y.: Large scale distributed deep networks. In: Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1. pp. 1223–1231. NIPS'12, Curran Associates Inc., USA (2012)
8. Dua, D., Graff, C.: UCI machine learning repository (2019), http://archive.ics.uci.edu/ml
9. European Commission: General data protection regulation (GDPR) (2018), https://ec.europa.eu/commission/priorities/justice-and-fundamental-rights/data-protection/2018-reform-eu-data-protection-rules

10. Hegedűs, I., Berta, Á., Kocsis, L., Benczúr, A.A., Jelasity, M.: Robust decentralized low-rank matrix decomposition. ACM Transactions on Intelligent Systems and Technology 7(4), 62:1–62:24 (May 2016)
11. Jelasity, M., Voulgaris, S., Guerraoui, R., Kermarrec, A.M., van Steen, M.: Gossip-based peer sampling. ACM Transactions on Computer Systems 25(3), 8 (Aug 2007)
12. Konecný, J., McMahan, H.B., Yu, F.X., Richtárik, P., Suresh, A.T., Bacon, D.: Federated learning: Strategies for improving communication efficiency. In: Private Multi-Party Machine Learning (NIPS 2016 Workshop) (2016)
13. McMahan, B., Moore, E., Ramage, D., Hampson, S., y Arcas, B.A.: Communication-efficient learning of deep networks from decentralized data. In: Singh, A., Zhu, J. (eds.) Proceedings of the 20th International Conference on Artificial Intelligence and Statistics. Proceedings of Machine Learning Research, vol. 54, pp. 1273–1282. PMLR, Fort Lauderdale, FL, USA (20–22 Apr 2017)
14. Montresor, A., Jelasity, M.: Peersim: A scalable P2P simulator. In: Proceedings of the 9th IEEE International Conference on Peer-to-Peer Computing (P2P 2009). pp. 99–100. IEEE, Seattle, Washington, USA (Sep 2009), extended abstract
15. Ormándi, R., Hegedűs, I., Jelasity, M.: Gossip learning with linear models on fully distributed data. Concurrency and Computation: Practice and Experience 25(4), 556–571 (2013)
16. Roverso, R., Dowling, J., Jelasity, M.: Through the wormhole: Low cost, fresh peer sampling for the internet. In: Proceedings of the 13th IEEE International Conference on Peer-to-Peer Computing (P2P 2013). IEEE (2013)
17. Wang, J., Cao, B., Yu, P.S., Sun, L., Bao, W., Zhu, X.: Deep learning towards mobile applications. In: IEEE 38th International Conference on Distributed Computing Systems (ICDCS). pp. 1385–1393 (Jul 2018)