Improving Gossip Learning via Limited Model Merging*

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

[1] University of Szeged, Szeged, Hungary
{danner,ihegedus,jelasity}@inf.u-szeged.hu
[2] ELKH SZTE Research Group on Artificial Intelligence, Szeged, Hungary

Abstract. Decentralized machine learning provides a unique opportunity to create data-driven applications without the need for large investments in centralized infrastructure. In our previous works, we introduced gossip learning for this purpose: models perform random walks in the network, and the nodes train the received models on the locally available data. We also proposed various improvements, like model sub-sampling, merging, and token-based flow control. Gossip learning is robust to failures, and does not require synchronization. Efficiency in terms of network bandwidth is also a major concern in the case of decentralized learning algorithms, especially when they are deployed in a network of IoT devices or smartphones. Here, we improve the model merging method to allow gossip learning to benefit more from token-based flow control. We experimentally evaluate our solution over several classification problems in simulations using an availability trace based on real-world smartphone measurements. Our results indicate that the improved variant significantly outperforms previously proposed solutions.

Keywords: Gossip learning · Decentralized machine learning · Model aggregation.

1 Introduction

The widespread presence of smart devices provides the possibility for numerous applications that use machine learning. The traditional approach to machine

* This version of the contribution has been accepted for publication in Proc. ICCCI 2023 after peer review but is not the Version of Record and does not reflect post-acceptance improvements, or any corrections. The Version of Record is available online at: http://dx.doi.org/10.1007/978-3-031-41774-0_28. Use of this Accepted Version is subject to the publisher's Accepted Manuscript terms of use https://www.springernature.com/gp/open-research/policies/accepted-manuscript-terms. This work was supported by the the European Union project RRF-2.3.1-21-2022-00004 within the framework of the Artificial Intelligence National Laboratory and project TKP2021-NVA-09, implemented with the support provided by the Ministry of Innovation and Technology of Hungary from the National Research, Development and Innovation Fund, financed under the TKP2021-NVA funding scheme.

learning is to collect the data at a central location for processing, but this can cause privacy concerns. This motivated a number of approaches to implement distributed machine learning algorithms. Perhaps the most notable approach is federated learning that is being used for mining data stored on smartphones without collecting it at a central location [12, 15]. Although the nodes perform the updates locally, there is still a need for a central server that aggregates and distributes the global model. Furthermore, the cost of such infrastructure can be prohibitive for startups or communities with limited resources.

To mitigate the problem of the requirement of a central infrastructure, a number of proposals have been made. Some of these utilize the blockchain infrastructure. Li et al. [14] proposed a blockchain-based, decentralized federated learning framework against malicious central servers or nodes. Ramanan and Nakayama [21] also proposed a blockchain-based federated learning framework that eliminates the role of a centralized aggregator. The selection of the device that updates a given chunk of the global model is based on the norm difference caused by the update. However, although the blockchain offers a number of benefits, blockchain-based distributed algorithms are not efficient.

Gossip learning [19] offers a more radical solution to the problem of central servers. Gossip learning is a fully distributed approach, not using any infrastructure (including blockchains) that avoids the collection of sensitive data, which remains on the devices instead. The devices communicate directly, sending their model to each other, and they train the received models on the locally available data. In addition to smartphones, this method has the potential to be employed on other platforms, such as smart metering or the Internet of Things. We presented a systematic comparison of federated learning and gossip learning in [11].

Gossip learning has been applied in a number of areas. For example, Guo et al. [10] proposed a gossip learning-aided user-centric online training framework to improve channel state information feedback performance and Belal et al. [3] proposed a decentralized recommender system based on gossip learning principles, which uses a personalized peer-sampling protocol and a model aggregation function that weights models by performance.

Giaretta and Girdzijauskas [9] assessed the applicability of gossip learning to real-world scenarios in general and introduced extensions that mitigate some of its limitations related to networks in which the node degree or the communication speed is correlated with the data distribution. Other improvements include the work of Onoszko et al. [18] that proposed a method for training personalized models in a decentralized setting with non-iid client data. Peers focus on communicating with neighbors that share similar data distributions, found based on training loss. Niwa et al. [17] extend Edge-Consensus Learning, an asynchronous decentralized DNN optimization algorithm, with the gradient modification of Stochastic Variance Reduction to improve performance on heterogeneous data.

There have also been a number of proposals to improve gossip learning specifically. Among these are model merging, model sub-sampling, and token-based flow control [11, 8]. However, these do not always work well together. In this paper, we propose a novel method for merging that improves its synergy with

token-based flow control, resulting in a significant speed-up even without subsampling. We evaluated our algorithm experimentally over several datasets and using a real-world smartphone availability trace (based on data collected by STUNner [4]).

The rest of the paper is structured as follows. In Section 2, we explain the concepts related to gossip learning. In Section 3, we describe our novel algorithm. In Section 4, we describe the experimental setup and discuss the simulation results. Finally, we conclude the paper in Section 5.

2 Background

Here, we introduce the necessary notations and concepts of gossip learning.

2.1 Supervised Classification

In the supervised learning problem, we aim to build a model based on a dataset $X = \{(x_1, y_1), \ldots, (x_n, y_n)\}$. Here, n is the size of the dataset (the number of samples). We assume that every sample x_i is a real valued feature vector of d elements ($x_i \in R^d$). In addition, for every example x_i, a class label y_i is also given that is an element of a discrete set K of possible class labels.

The learning task can be formulated as an optimization problem. The aim of the learning procedure is to find the parameters w of a given hypothesis function $h_w : R^d \to K$ that minimizes the objective function

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

This objective function is the average of the losses ℓ computed on the training samples. To improve generalization, a regularization term is also often added, where λ is the regularization coefficient. A common solution for this optimization problem is the Gradient Descent (GD) method, where we iteratively update the parameters of the model, based on the partial derivative of the objective function.

$$w_{t+1} = w_t - \frac{\partial J(w)}{\partial w}(w_t) \tag{2}$$

Stochastic Gradient Descent (SGD) [6] approximates this update using the derivative of the loss of only one sample at a time. It iteratively selects a training example, sampled uniform randomly, and performs the parameter update of the model using the update rule

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

where η_t is called the learning rate.

Algorithm 1 Basic version of gossip learning

```
1: (x, y) ← local sample
2: (w, t) ← initialize()                                   ▷ local model
3: loop
4:     wait(Δ)
5:     p ← selectPeer()                                    ▷ returns a random online neighbor
6:     send (w, t) to p
7: end loop
8:
9: procedure onReceiveModel(w_r, t_r)
10:    (w, t) ← (w_r, t_r)
11:    (w, t) ← update((w, t), (x, y))                     ▷ the model is trained and t is incremented
12: end procedure
```

Logistic regression [5] is a linear model that specifies the hypothesis function as

$$h_{(w,b)}(x_i) = \frac{1}{1 + e^{(w^T x_i + b)}}, \tag{4}$$

where b is an additional model parameter, called bias, and $y_i \in \{0, 1\}$. The loss function is

$$\ell(h_{(w,b)}(x_i), y_i) = -y_i \ln h_{(w,b)}(x_i) - (1 - y_i) \ln(1 - h_{(w,b)}(x_i)). \tag{5}$$

2.2 Gossip Learning

In traditional machine learning, the model is trained on one machine or on a cluster of servers that stores the model and the dataset and performs the model updates. But in gossip learning [19], we have a network of computational units that are typically connected via the Internet. The dataset is distributed on these devices horizontally, that is, every node in this network holds only a few or maybe just one sample from the dataset. Models perform random walks (series of random steps) in this network, and when a node receives a model, it updates it by applying the SGD method using the local samples. More precisely, each node in the network first initializes a new model, and stores it locally. After that it periodically sends its local model to one of its neighbors in the network. When a node receives a model, it updates and stores this model locally.

The neighborhood management is provided by a peer sampling service. If this service can provide a node picked uniformly at random from the network, the model will be updated on training examples sampled uniformly at random.

An advantage of this decentralized approach is that, due to every node storing a model locally, they can predict the label of unseen examples without any communication cost.

The basic version of gossip learning can be seen in Algorithm 1. Every node holds only one data sample (x, y) and has a model (w, t) locally, where the model age t is the number of times the model has been updated. Δ denotes the gossip cycle length.

We call this communication pattern *proactive* because messages are sent based on a gossip cycle, independently of other events. In this proactive scheme, the number of the randomly walking models in the network approximately equals the number of online nodes. In the case of a linear model, the model size (the number of parameters) is the same as the size of an example, plus an extra bias term. In the case of more complex models, like neural networks, the model can be much larger.

Various techniques can be employed to improve the above algorithm, as listed below.

Token-based flow-control The communication pattern can be adjusted so that models perform "hot potato"-like random walks, that is, models do not wait for the clock at each node [8]. This technique is detailed in Section 2.3.

Model merging Instead of overriding the local model by the received one, the average of the model parameters is stored. One way to do this is to weight the models by their age [11].

Model partitioning This is a sub-sampling technique where there is a predefined partitioning of the model parameters; instead of sending the whole model to a neighbor, only a uniformly random partition is sent [11]. This reduces the communication cost of each message, meaning we can send messages more frequently. When such a partial model is received, it is merged with the corresponding parameters, leaving the rest unchanged (but the whole model is updated afterwards using the local samples). Each partition has its own age, which is used during weighted model merging.

Transfer learning We can use a large, pre-trained model (that was trained on a related, but different task) as a feature extractor [22]. This can result in a more useful and/or smaller feature set, the latter reducing communication costs. This can be the equivalent of training only the last layer of a deep neural network.

2.3 Token Gossip Learning

The proactive approach, that is, periodically sending the local model is suboptimal when the model transfer time is much shorter than the cycle length, that is, when the allowed average bandwidth utilization is much smaller than the maximum available bandwidth. This is because a lot of time is wasted between receiving a model and forwarding it to another node. The purely reactive approach, that is, forwarding a model immediately after receiving (and updating) it, is prone to extinction due to message loss. (Even reliable protocols like TCP can't guard against churn, the prolonged unavailability of a node.) The solution is the token account algorithm [8], a hybrid approach that enables a small number of models to travel (and learn) quickly in the network, and manages the number of such models implicitly, in an emergent way.

Similarly to the token bucket algorithm [20], a token is granted periodically at each node which can be later spent on outgoing network communication, and there is a cap on the number of tokens that can be accumulated. However,

Algorithm 2 Token gossip learning

```
1: (x, y) ← local sample
2: (w, t) ← initialize()                                  ▷ can include initial update
3: a ← 0                                                          ▷ number of tokens
4: loop
5:     wait(Δ)
6:     do with probability proactive(a)                          ▷ randomized branching
7:         p ← selectPeer()
8:         send (w, t) to p
9:     else
10:        a ← a + 1                  ▷ we did not spend the token so it accumulates
11:    end do
12: end loop
13:
14: procedure onReceiveModel(w_r, t_r)
15:    (w, t) ← merge((w, t), (w_r, t_r))
16:    (w, t) ← update((w, t), (x, y))
17:    x ← randRound(reactive(a))  ▷ randRound(x) rounds up with probability {x}
18:    a ← a − x                                                  ▷ we spend x tokens
19:    for i ← 1 to x do
20:        p ← selectPeer()
21:        send (w, t) to p                          ▷ queued for sequential sending
22:    end for
23: end procedure
```

here, message sending is not completely reactive. Messages can also be sent proactively (spending a newly granted token immediately), to avoid starvation. Moreover, to preemptively mitigate the impact of message loss, multiple copies of a received model may be transmitted to various neighbors. If the number of tokens is approaching the capacity, it is an indication that there might be too few random walks in the network, therefore the algorithm becomes more inclined to send a proactive message (starting a new random walk) or multiple reactive messages (duplicating a random walk). The exact behavior is defined by functions $\textsc{proactive}(a)$ and $\textsc{reactive}(a)$, where a is the number of tokens in the node's account. $\textsc{proactive}(a)$ returns the probability of sending a proactive message, and $\textsc{reactive}(a)$ returns the number of reactive messages to be sent. $\textsc{reactive}(a)$ is allowed to return a non-integer; we round it up with the probability equaling the fractional part, and round it down otherwise.

Here, we shall use the (slightly simplified) randomized strategy [8] that defines these functions as

$$\textsc{proactive}(a) = \begin{cases} 0 & \text{if } \; a < A-1 \\ \dfrac{a-A+1}{C-A+1} & \text{if } \; a \in [A-1, C] \end{cases}$$

and $\textsc{reactive}(a) = a/A$. C is the capacity of the token account, and A influences the size of the token reserve the algorithm tries to maintain. The algorithm

guarantees that a node will not send more than $\lceil t/\Delta \rceil + C$ messages within a period of time t. The pseudocode for the token account algorithm as applied to gossip learning is shown in Algorithm 2, also including model merging. This technique can result in a much faster growth of model age, since the (on average) $\Delta/2$ waiting period between the steps of the random walk is eliminated, leaving only the transfer time.

There is also a variant of this algorithm for use with partitioned model sampling [11]. It uses a separate token account for each partition, and reactive messages contain the same partition as the received one.

3 Limited Merging

Previous approaches to model merging used unweighted [19] or weighted [11] averaging, where the weights are the model ages. More precisely, for two models (w_a, t_a) and (w_b, t_b) the merged model is given by

$$\frac{t_a \cdot w_a + t_b \cdot w_b}{t_a + t_b}.$$

The age of the resulting model is $\max(t_a, t_b)$.

There are a number of situations, however, when one of the models has significantly more fresh updates. For example, one of the models might have collected many updates during a "hot potato" run due to the token account algorithm. Another possibility is that one of the models was not updated for some time due to its node having been offline. In such situations, it is a problem if unweighted merging is used, because this essentially halves the contribution of the freshly updated model. Weighted averaging does not solve this problem, because after a certain amount of time the age of both of the models will be so large that weighted averaging will effectively work as unweighted averaging due to the relative age difference becoming too small.

To deal with this problem, we propose limited merging. Here, when the age difference of the models to be merged is above a threshold L, we simply take the model with the higher number of updates and throw away the other one, instead of performing weighted averaging. The pseudocode is shown in Algorithm 3. In the case of partitioned models, this rule can be applied to the received partition and its counterpart based on their age.

4 Experiments

We performed the experimental evaluation using simulations[3] building upon PeerSim [16] to measure the gain in convergence speed resulting from our novel algorithm. In this section, we describe the datasets, system model, availability trace, and hyperparameters we used, then present our results.

[3] https://github.com/ormandi/Gossip-Learning-Framework/tree/privacy

Algorithm 3 Limited merging

```
1: procedure MERGE((w_a, t_a), (w_b, t_b))
2:     if t_a > t_b + L then
3:         w ← w_a
4:     else if t_b > t_a + L then
5:         w ← w_b
6:     else
7:         w ← (t_a · w_a + t_b · w_b)/(t_a + t_b)
8:     end if
9:     t ← max(t_a, t_b)
10:    return (w, t)
11: end procedure
```

4.1 Datasets

We used three different classification datasets in our experiments: the Pendigits and HAR datasets, and a transformed version of the MNIST dataset. The Pendigits [2] and the MNIST [13] samples represent hand-written numbers from [0-9], forming 10 classes. The Pendigits dataset represents the numbers using 16 attributes, while MNIST contains images of 28×28 pixels. In a previous study [7], we performed transfer learning to extract new, high-quality features from the MNIST database using a CNN model that was trained over another dataset: Fashion-MNIST. The Fashion-MNIST [24] dataset has the same properties as MNIST, but it contains images of clothes and shoes instead of digits. We assume that the nodes have downloaded this pre-trained model and use it to transform the local samples. The feature set was compressed to 78 attributes using Gaussian Random Projection, to reduce the bandwidth consumption of gossip learning through the smaller models. In the HAR (Human Activity Recognition Using Smartphones) [1] dataset, the labels are walking, walking upstairs, walking downstairs, sitting, standing and lying, and the attributes are high level features extracted from smartphone sensor measurement series (acceleration, gyroscope and angular velocity). The features of all the datasets were standardized, that is, scaled and shifted to have a mean of 0 and a standard deviation of 1. Some important statistics of the datasets can be seen in Table 1.

Table 1. Data set properties. Note that the transformed version of MNIST, that we used in the experiments, has only 78 features.

	Pendigits	HAR	MNIST	F-MNIST
Training set size	7494	7352	60000	60000
Test set size	3498	2947	10000	10000
Number of features	16	561	784*	784
Number of classes	10	6	10	10
Class-label distribution	$\approx$ uniform	$\approx$ uniform	$\approx$ uniform	$\approx$ uniform

4.2 System model

We consider a network of unsynchronized nodes. The size of the network is assumed to be identical to the training set size of the given database. Thus, each node has a unique training sample. Each node also has a random list of 20 nodes it can connect to, as this provides a favorable mixing time. (In practice, such a list can be obtained and maintained by a decentralized peer sampling service like Newscast [23].) In the churn scenario, we assume that the nodes may go offline at any time, and later may come back online (with their state intact). For a message to be successfully delivered we require that both the sender and the receiver remain online for the duration of the transfer. The allowed average bandwidth utilization was set so that a continuously online node can send 1000 full models in 24 hours. We assumed that nodes communicate during 1% of their online time, that is, the allowed average bandwidth utilization is much lower than the allowed maximum bandwidth utilization, enabling bursts. To model a platform where different learning tasks are solved one after the other, we simulated 48 hours, but performed learning only during the second 24 hours; this is to ensure a realistic distribution of token counts in the network at the beginning of the learning task.

4.3 Smartphone Trace

In our experiments involving churn, we used a smartphone availability trace collected from 1191 users by an application called STUNner [4], which monitors the NAT (network address translation) type, bandwidth, battery level, and charging status. The time series was split into 40,658 2-day segments (with a one-day overlap) based on the UTC time of day. By assigning one of these segments to each simulated node, it becomes possible to simulate a virtual 48-hour period. The segments are randomly sampled without replacement, but whenever the pool of segments runs out, the pool is reset. We define a device to be available after it has been on a charger as well as connected to the Internet (with a bandwidth of at least 1 Mbit/s) for at least one minute. This means we consider a user-friendly scenario in which battery power is not used at all.

Some important properties of this trace are shown in Figure 1. On the right, the blue and red bars represent the ratio of the nodes that joined or left the network (at least once), respectively, in a given hour. Observe that usually only about 20% of the nodes are online. The average online session length is 81.37 minutes.

4.4 Hyperparameters

Over a given dataset, we trained logistic regression models for each class separately, embedded in a one-vs-all meta-classifier. We set the cycle length Δ to one thousandth day (86.4 seconds). The algorithm variants using partitioned subsampling sent a random partition containing 10% of the model parameters

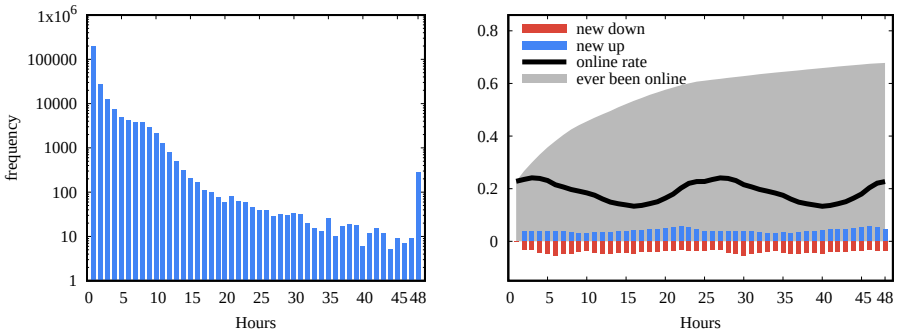

Fig. 1. Histogram of online session lengths (left) and device churn (right).

at a time, and the gossip cycle time was proportionally shorter so that the overall bandwidth utilization is the same. For token gossip learning, we used the randomized strategy with parameters $A = 10$ and $C = 20$. We used a dynamic learning rate $\eta_t = \eta/t$, where t is the model age (not the elapsed wall-clock time). The regularization coefficient was set to $\lambda = 1/\eta$. In the case of the baseline algorithms, we chose η from the set of values $\{10^0, ..., 10^5\}$ so as to minimize the classification error at the end of the churn-based simulation.

The hyperparameters of the novel variants (η and L) were also optimized using a simple grid search. For the error-free scenario, we used the same hyperparameters that were used in the churn scenario. We note here that if there is more than one sample in a node then the parameter L should be scaled proportionally because the age of the models will be scaled as well.

4.5 Results

Our results are shown in Figure 2. The plots show the average ratio of misclassified test samples over the models of the online nodes. The horizontal axis is time (measured in cycles), which is also an upper bound on the total communication cost.

We can see that the token gossip learning method benefits greatly from the novel merging method, as the models performing hot potato walks are no longer held back by the frequent merging with inferior models. The combination of using the token account technique and limited merging outperformed the other variations by a large margin across all the datasets. (Note the logarithmic horizontal scale.) By comparing the two columns in Figure 2, we can see that the performance is fairly robust to node failures.

When using limited merging, the algorithm variants that use partitioned models are mostly inferior to the variants communicating the entire model. This can be seen in Figure 3. Note that this is not due to the partitioned variants communicating less: we specifically decreased the gossip cycle to equalize expected communication.

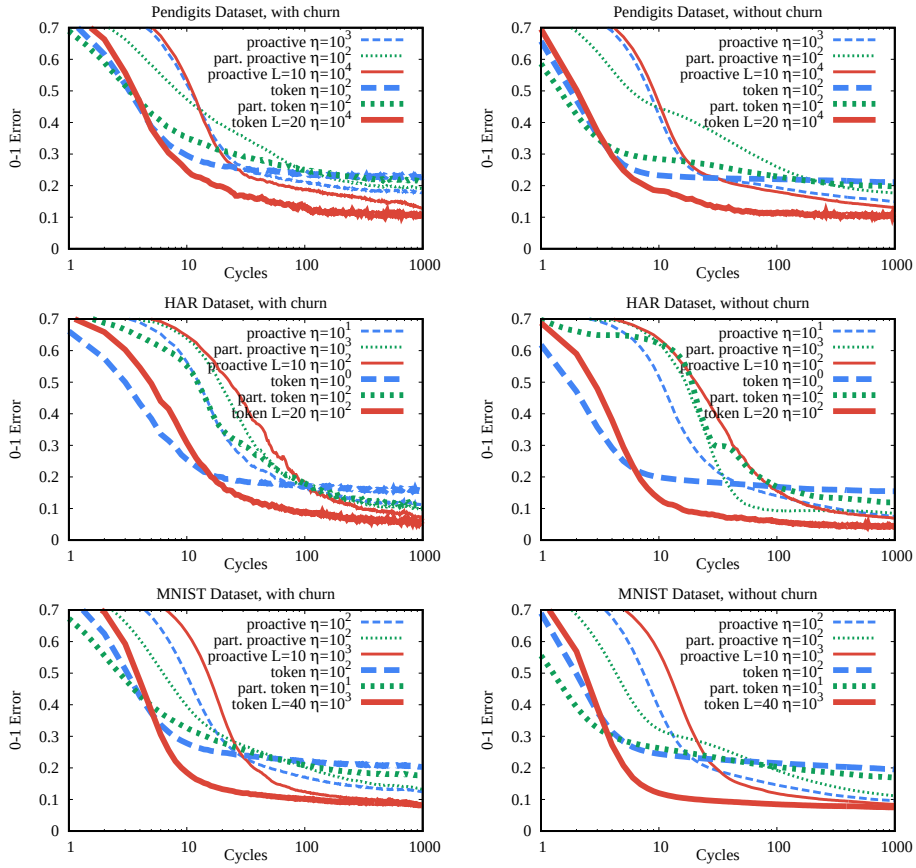

Fig. 2. Classification error of the various algorithm variants as a function of time, with trace-based churn (left) and without failures (right). "part." denotes partitioned.

Figure 3 also shows the sensitivity of token gossip learning to the hyperparameter settings. We can see that the sensitivity depends on the complexity of the dataset, but in general, it is advisable to pay attention to hyperparameter optimization.

5 Conclusions

Gossip learning enables the training of machine learning models over a network of unreliable devices without the need for a costly central server. When training large models, it becomes important to make the algorithm efficient in terms of network communication. When the nodes are allowed to communicate in bursts, token gossip learning can be used to speed up learning while keeping the total bandwidth consumption of each node unchanged. In this paper, we proposed a novel method for model aggregation that vastly improved the performance of token gossip learning, as we demonstrated on several learning tasks.

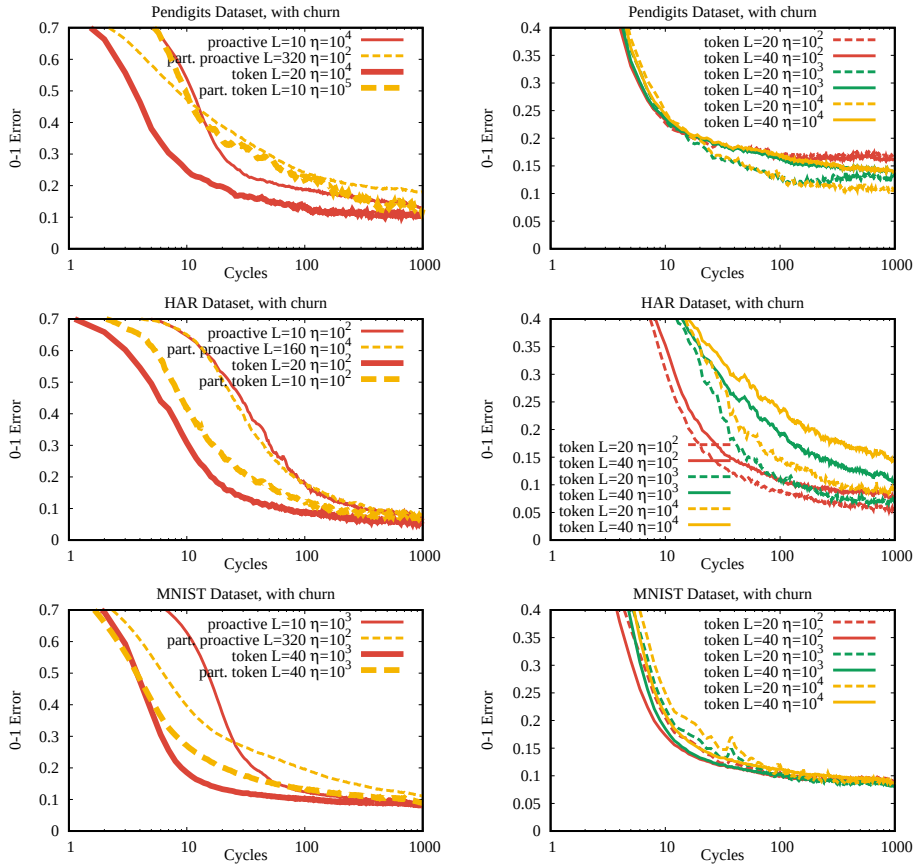

Fig. 3. Merge-limited gossip learning with and without partitioned subsampling (left) and the effect of different hyperparameter settings (right).

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: Esann. vol. 3, p. 3 (2013)
2. Bache, K., Lichman, M.: UCI machine learning repository (2013)
3. Belal, Y., Bellet, A., Mokhtar, S.B., Nitu, V.: PEPPER: Empowering user-centric recommender systems over gossip learning. Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies **6**(3), 1–27 (2022)
4. 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). https://doi.org/10.1109/P2P.2014.6934317
5. Bishop, C.M., Nasrabadi, N.M.: Pattern recognition and machine learning, vol. 4. Springer (2006)
6. Bottou, L.: Stochastic gradient descent tricks. In: Montavon, G., Orr, G.B., Müller, K.R. (eds.) Neural Networks: Tricks of the Trade, Lecture Notes in

Computer Science, vol. 7700, pp. 421–436. Springer Berlin Heidelberg (2012). https://doi.org/10.1007/978-3-642-35289-8_25
7. Danner, G., Hegedűs, I., Jelasity, M.: Decentralized machine learning using compressed push-pull averaging. In: Proceedings of the 1st International Workshop on Distributed Infrastructure for Common Good. pp. 31–36 (2020)
8. 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). https://doi.org/10.1109/ICDCS.2018.00090
9. Giaretta, L., Girdzijauskas, Š.: Gossip learning: Off the beaten path. In: 2019 IEEE International Conference on Big Data (Big Data). pp. 1117–1124. IEEE (2019)
10. Guo, J., Zuo, Y., Wen, C.K., Jin, S.: User-centric online gossip training for autoencoder-based CSI feedback. IEEE Journal of Selected Topics in Signal Processing **16**(3), 559–572 (2022)
11. Hegedűs, I., Danner, G., Jelasity, M.: Decentralized learning works: An empirical comparison of gossip learning and federated learning. Journal of Parallel and Distributed Computing **148**, 109–124 (2021)
12. Konečnỳ, J., McMahan, H.B., Yu, F.X., Richtárik, P., Suresh, A.T., Bacon, D.: Federated learning: Strategies for improving communication efficiency. arXiv preprint arXiv:1610.05492 (2016)
13. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proc. of the IEEE **86**(11), 2278–2324 (Nov 1998), http://yann.lecun.com/exdb/publis/pdf/lecun-98.pdf
14. Li, Y., Chen, C., Liu, N., Huang, H., Zheng, Z., Yan, Q.: A blockchain-based decentralized federated learning framework with committee consensus. IEEE Network **35**(1), 234–241 (2020)
15. McMahan, B., Moore, E., Ramage, D., Hampson, S., y Arcas, B.A.: Communication-efficient learning of deep networks from decentralized data. In: Artificial intelligence and statistics. pp. 1273–1282. PMLR (2017)
16. Montresor, A., Jelasity, M.: PeerSim: A scalable P2P simulator. In: Proc. 9th IEEE Intl. Conf. Peer-to-Peer Computing (P2P 2009). pp. 99–100. IEEE, Seattle, Washington, USA (Sep 2009). https://doi.org/10.1109/P2P.2009.5284506, extended abstract
17. Niwa, K., Zhang, G., Kleijn, W.B., Harada, N., Sawada, H., Fujino, A.: Asynchronous decentralized optimization with implicit stochastic variance reduction. In: International Conference on Machine Learning. pp. 8195–8204. PMLR (2021)
18. Onoszko, N., Karlsson, G., Mogren, O., Zec, E.L.: Decentralized federated learning of deep neural networks on non-iid data. arXiv preprint arXiv:2107.08517 (2021)
19. 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)
20. Partridge, C.: Gigabit networking. Addison-Wesley Professional (1994)
21. Ramanan, P., Nakayama, K.: Baffle: Blockchain based aggregator free federated learning. In: 2020 IEEE international conference on blockchain (Blockchain). pp. 72–81. IEEE (2020)
22. Shin, H., Roth, H.R., Gao, M., Lu, L., Xu, Z., Nogues, I., Yao, J., Mollura, D., Summers, R.M.: Deep convolutional neural networks for computer-aided detection: CNN architectures, dataset characteristics and transfer learning. IEEE Transactions on Medical Imaging **35**(5), 1285–1298 (2016)
23. Tölgyesi, N., Jelasity, M.: Adaptive peer sampling with newscast. In: Euro-Par. vol. 5704, pp. 523–534. Springer (2009)

24. Xiao, H., Rasul, K., Vollgraf, R.: Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. arXiv preprint arXiv:1708.07747 (2017)