A strategy to the reduction of communication overhead and overfitting in Federated Learning

Alex Barros[1], **Denis Rosário**[1], **Eduardo Cerqueira**[1], **Nelson L. S. da Fonseca**[2]

[1]Federal University of Pará (UFPA)
Belém – PA – Brazil

[2]Institute of Computing
University of Campinas
Campinas, Brazil

{alexbarros, denis, cerqueira}@ufpa.br, nfonseca@ic.unicamp.br

Abstract. *Federated learning (FL) is a framework to train machine learning models using decentralized data, especially unbalanced and non-iid. Adaptive methods can be used to accelerate convergence, reducing the number of rounds of local computation and communication to a centralized server. This paper proposes an adaptive controller to adapt the number of epochs needed that employs Poisson distribution to avoid overfitting of the aggregated model, promoting fast convergence. Our results indicate that increasing the local update of the model should be avoided, but yet some complementary mechanism is needed to model performance. We evaluate the impact of an increasing number of epochs of FedAVG and FedADAM.*

1. Introduction

Emerging technologies, such as augmented reality, the Internet of Things (IoT), and social networking applications, have led to unprecedented growth in the volumes of data generated daily. In line with that, organizations must rethink their data governance, retention, and data privacy policies. For instance, the International Data Corporation (IDC) [Mukherjee and Rojas 2020] predicted that, by 2025, 79ZB of data will be created by billions of IoT devices. On the other hand, data privacy has been the focus of attention of several governments which makes the centralization of data in cloud servers a real concern. Some legislation such as the U.S. Consumer Privacy Bill of Rights, the European Commission's General Data Protection Regulation (GDPR), and the General Law of Data Protection (LGPD) in Brazil have been designed to protect users' privacy.

Large propagation delays to the cloud server and the lack of capacity to provide the requirements of delay-sensitive applications have motivated the deployment of edge computing which aims at bringing cloud services to the network edge close to the end-user [Abdulrahman et al. 2021]. Federated Learning (FL) paradigm has recently been proposed to enable distributed machine learning process while preserving the privacy of data by processing them at the site (client) they are usually generated. the parameters of locally trained models are then sent over the network to a central server which aggregates the value of the several parameters received and return to the clients a consolidated model, which then will be trained again with local data by the participant clients in a federation [McMahan et al. 2017] and [McMahan et al. 2017].

FL can be realized mainly in two different settings namely, cross-silo and cross-device [Kairouz et al. 2021]. Cross-silo FL usually occurs by the exchange of machine learning models between institutions assuming reliable network connection and a more controlled environment. For example, training a model for medical treatment using patient records from multiple hospitals [Yu et al. 2020]. Cross-device FL involves thousands of edge-devices participating in the federation. In this setting, usually, a small fraction of clients participate in each round of model training, clients cannot maintain state across rounds most of the time, and there are constraints related to network communication, mobility, and edge computing [Reddi et al. 2020]. For example, training a next-word prediction model on texts typed by users into their smartphones or image-classification task of driving signals. There are many others related to keyboard prediction such as Federated Recurrent Neural Network (RNN) that increases next-word prediction accuracy by +24% [Hard et al. 2019], Emoji prediction [Ramaswamy et al. 2019], Action prediction, and discovering new words that people are typing [Chen et al. 2019].

In this paper, we investigate the last scenario for which communication-efficient protocols still need to be developed. In a cross-device setting, each client has a local training dataset that is never uploaded to the server [McMahan et al. 2017]. Instead, each client computes a local update of the model parameter that is communicated to the server. This reduces privacy and security risks by limiting the attack surface to individual devices, rather than all the devices and the cloud.On the other hand, standard optimization methods highly used in machine learning, such as distributed Stochastic Gradient Descent (SGD), are may incur large communication overhead if frequently communicated to the central server. Alternatively, clients perform multiple local updates before communicating the model parameters to the server. This can significantly reduce the amount of communication required to train a model [Reddi et al. 2020]. Federated averaging (FedAVG) is a baseline algorithm proposed in [McMahan et al. 2017] to aggregate at the central server parameters coming from several clients. Although highly used, convergence issues exist when employing this algorithm [Karimireddy et al. 2021].

Moreover, we explored the FedOPT, a flexible algorithmic framework that allows clients and server to choose optimization methods other than the SGD method in FedAVG [Reddi et al. 2020]. However, only a few works have explored this framework and performed an empirical evaluation of its effectiveness considering communication efficiency issues [Hsu et al. 2019] [McMahan and Streeter 2010].

The main contributions of this paper are:

1. An evaluation of how local adaptation can lead to better accuracy of a model considering communication restrictions and overfitting of machine learning models. For that, we performed an empirical evaluation of FedAVG and FedOPT using ADAM optimizer.
2. Demonstration based on numerical results that multiple times of local updates can be used to improve the accuracy of training, due to the reduced number of rounds required to achieve a desired accuracy or loss, especially in scenarios with network constraints and heterogeneity of clients' resources and data distribution.
3. A dynamic adaptation to include a variable number of epochs at each round to find the better trade-off between high accuracy, less round communication, and avoid overfitting.

The remainder of this paper is structured as follows: Section 2 outlines an overview of FL and its main challenges. Section 3 outlines the strategies to improve communication efficiency adopted by recently related works. Section 4 introduces our experiments varying some parameters of federated algorithms implementations and section 5 presents our results. Section 6 presents the concluding remarks and future works.

2. Federated Learning challenges

The main advantage of FL is to train a machine learning model in a distributed way, protecting the privacy of local data by communicating only the model's parameters to a central server, and not the dataset. The edge devices are called clients, which are the data owner and responsible for storing local data observations. A coordinating server (often called parameter server) aggregates the local parameters from all the clients, derives an updated model, and shares this model with the participating clients to benefit from their learning experience and to enable them to pursue their local training in future iterations [Wahab et al. 2021]. FL has faces several challenges related to privacy, communication, latency, data heterogeneity, and connectivity.

Privacy is one of the most relevant features of FL. In FL, raw local data never leaves the user's device since the training is done locally on each device. However, it could be a target of inference attacks that aim to infer sensitive information from user's training data [Melis et al. 2018]. As a huge number of devices can be used in the training, this risk may be of concern.

Communication overhead is reduced in FL since no raw data need to be transferred to a central server. However, since the machine learning models are trained collaboratively, updates on the parameters of the models need to be communicated between to the server in several iterations, which poses additional communication costs [Wahab et al. 2021]. The naive strategy to reduce communication in federated optimization is limiting the number of devices involved. Alternatively, reducing the number of communication rounds, and reducing the size of messages sent over the network. The present work focuses on the reduction of communication rounds [Yu et al. 2020] and [Wang et al. 2021].

Latency is expected to decrease as the models are trained locally on the edge/end devices and can be updated faster and have greater generalization power. Such generalization depends on the statistical heterogeneity and their usage patterns. The local dataset of a client is not expected to be representative of the overall data distribution [Wahab et al. 2021] because clients use their own services or applications in different degrees. As a consequence, local datasets have different sizes and content distribution. Another issue is related to connectivity, client devices are frequently offline or on slow or expensive connections caused by the channel heterogeneity when on move. The client selection process can be biased toward certain conditions (e.g., local time zone, the device being charged or not, etc.) [Wahab et al. 2021]

[Wahab et al. 2021] divided the main challenges in FL into six areas and observed that the statistical and communication efficiency challenges have been the most investigated, accounting for 25% of the surveyed papers, followed by privacy concerns with 19%, client selection and scheduling with 14%, the security concerns with 10%, and, finally the service pricing with 7%.

The general idea of FL is depicted in Figure 1. It consists of one central server called parameter server and a set of N clients, each having its local dataset. At the beginning of each federated training iteration, a subset C $\subseteq$ N of clients is chosen to receive the current global state of the shared model in terms of model weights. This model is broadcasted to clients (step 1). Each client uses its CPU and energy resources to carry out local computations on its dataset based on the shared parameters (step 2). Clients then send the model updates (step 3) to the parameter server which applies these updates to its current global model to generate a new one (step 4).

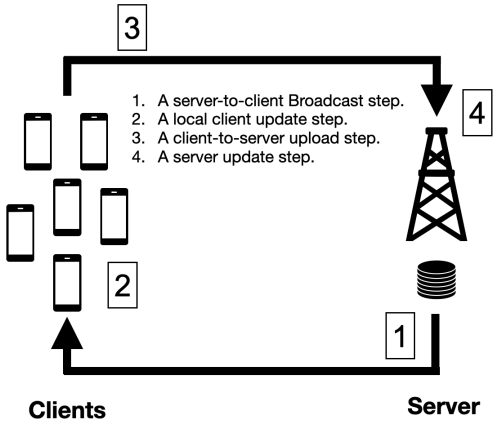

Figure 1. The overview of four main steps of FL Training Process

This process is repeated over several iterations (sometimes referred to as epochs) until the global model reaches a certain accuracy level determined by the parameter server. In summary, an FL scenario consists of two main phases, i.e., local update and global aggregation. The local update phase refers to the process of computing the gradient descents by the client devices to minimize the underlying loss function for their local data. Global aggregation includes the steps of collecting the updated model parameters by the server from the different client devices, aggregating these parameters, and then sending back the aggregate parameters to the clients to be used in their next training iteration.

In terms of optimization research, there are opportunities in client and server optimization. The former is related to the process of local updates/training while the later is related to the process aggregating the model parameters sent by clients to produce an enhanced model. FL needs some orchestration back and forth between the server and the devices to train and evaluate a model, there are many architecture and strategies related to this orchestration. The initial model can be either initialized randomly or can be pre-trained. This process happens hundreds or even thousands of times in model training.

The service provider is interested in the data that the device has. The initial model can be initialized randomly or we can pre-trained it. This process is going to happen hundreds or maybe thousands of times. Both the initial model and how the model evolves are important and must get attention from the management entity that controls all this process.

To determine the optimal set of parameters that fits the training data, the training model has to optimize a loss (objective) function, which penalizes the model when it

produces an inaccurate label on a data point. Stochastic gradient descent is the backbone of ML and for large datasets, it is implemented distributed where the dataset is shuffled and split equally across worker nodes. Ideally, if we have more nodes, we are processing more data per iteration and we expect to increase the speed to get a target accuracy, but in reality, it is not so easy to achieve such speed due to issues related to synchronization and communication delays that increase with the number of clients. The problems related to synchronization delays are related to workers with less computing power as demonstrated in [Dutta et al. 2018]. Communication delay is the time taken to aggregate the gradients, update them, and send the updated model back to the workers/clients.

One solution to overcome this communication delay is called local update SGD, basically, the workers perform more local updates instead of computing only one update and send the model to the server aggregates. This reduces the frequency of communication and makes the derivation of a single model more efficient. However, clients are not homogeneous as expected and they are not available for training all the time. Then, just add more clients does not bring the process to convergence. If too many clients drop off the training process can become unstable.

Finally, we highlight the challenges as data are heterogeneous across devices, both in size and composition. For example, if two devices are in a different part of the world they might have a different pattern of application use, and the data generated can be very different. The computation resource can also greatly vary between devices (mobile devices, tablets, vehicles) and they can be geographically distributed. Further, only a sub-set of clients at a given time. Moreover, the choice of the best clients can be stated as an optimization problem. In the following section, we summarize some related research that addressed the problem of communication efficiency in FL. Many of than explored alternative algorithms to increase the accuracy and reduce the loss considering the heterogeneity.

3. Related Works

Several works have been proposed to improve the FL. Some of them propose customization of both server and client. [Li et al. 2020] proposed FedProx, including a component to represent the statistical heterogeneity across client devices. The authors added a proximal term to the local training subproblem on each client device. This adaptation allows the local updates to get closer to the initial global model, limiting the influence of each local model update on the global model. This adaptation is proposed to improve the convergence when dealing with statistically heterogeneous data, and when all the devices are weighted equally in the global aggregation phase, disregarding the differences in the device capabilities (e.g., hardware).

[Reisizadeh et al. 2020] proposed FedPAQ focusing on periodic averaging of the local model updates. Instead of synchronizing their model updates with the server at each iteration, it enables clients to carry out multiple local updates on the model prior to sharing the updates with the server. The clients should be chosen to participate in the training at each iteration on the basis of factors such as the device connectivity to a free wireless network, idle, and reachable to a base station. No evaluation of overfitting or fairness was performed in this work.

[So et al. 2021] introduced a communication and security-oriented aggregation technique based on a multi-group circular strategy. Turbo-Aggregate divided the clients

into multiple groups and at each iteration the clients belonging to one group transmit the aggregated model updates of all clients in the previous groups and the local model updates of the current group to the clients of the next group. It includes a security component for sharing mechanisms and preserve the privacy of clients' data. This proposition does not adapt to new users that join the network and could be extended to a self-configurable protocol.

In addition, [Wang et al. 2020] proposed the FedMA, a statistics-oriented aggregation technique which uses permutation invariance of the neurons in the neural network model before performing aggregation. It aims to adapt the model size employing a Bayesian non-parametric mechanism which allows it to adjust the size of the central model to the heterogeneity of data distribution. Unfortunately, FedMA can be vulnerable to model poisoning attacks and some additional mechanisms should be added to improve security.

The problem of 'client-drift' is investigated in [Karimireddy et al. 2021], when data are heterogeneous (non-iid), resulting in unstable and slow convergence. The authors proposed a new algorithm called Stochastic Controlled Averaging algorithm (SCAFFOLD), that uses control variate to reduces the effect of data heterogeneity and client sampling. This work also indicates that most of the methods benefit from an increasing number of epochs but they didn't evaluate potential overfitting.

The non-IID data distribution is a consequence of the inherent heterogeneity in the local data generated across clients' device. Since each local device records the activities of its owner, data across devices tend to have different sizes, features, and target classes distribution. Naturally, the local data of one single client cannot be considered to be representative of the overall data distribution [Wahab et al. 2021].

Federated optimization methods that perform local updating can significantly reduce communication rounds needed for convergence. However, heterogeneity can potentially lead to slower convergence, reduced stability, or divergence. In our experiments, we evaluate the impact of the increasing number of epochs (local updates) of FedAVG and FedADAM using the same protocol proposed in [Reddi et al. 2020]. With the proven impact and possible overfitting of the model, we propose an adaptive controller to vary the number of epochs using a Poisson distribution to avoid overfitting.

4. Evaluation

In this section, We present an empirical comparison between FedAVG and FedADAM in different scenarios involving different local update settings. The aim is to assess how the number of epochs(local updates) can impact the convergence, especially in cross-device settings. To accomplish this, we conduct simulations employing the dataset EMNIST. It consists of images of digits and upper and lower case English characters, with 62 total classes. The federated version of EMNIST [Caldas et al. 2019] partitions the digits by author. The dataset has natural heterogeneity stemming from the writing style of each person. We train a Convolutional Network for character recognition. The network has two convolutional layers (with 3 x 3 kernels), max pooling, and dropout, followed by a 128 unit dense layer similar as used in [Reddi et al. 2020].

The implementation was based on TensorFlow Federated and we should highlight some features as the clients are sampled uniformly at random, without replacement in

a given round, but with replacement across rounds. Second, instead of performing K training steps per client, we perform E epochs of training over each client's dataset. Last, to account for varying numbers of gradient steps per client, we weight the average of the client outputs by each client's number of training samples. This follows the approach adopted in [McMahan et al. 2017].

An implementation based on FedOpt was used. For the FedAVG simulations, we used ClientOpt and ServerOpt with SGD with learning rates 0.1 and 1.0 for client and server, respectively; The batch size was defined in 20 for EMNIST. For FedAdam, the client learning rate was 0.03 and the server was at 0.003. Other parameters were defined as proposed in [Reddi et al. 2020], which showed an extensive evaluation and grid-search of the best parameter values to minimize the average training loss over the last 100 rounds of training.

The simulations also collected some validation metrics. They were measured on a validation set throughout training using the entire test set. The number of communication rounds is the main parameter to evaluate how fast is the convergence, but a more complex and realistic scenario can be defined.

We perform 1500 rounds of training and Figure 2 shows how the increase in local updates can improve the speed of convergence. The metric loss and accuracy improve needing fewer communication rounds as demonstrated in Table 1 .

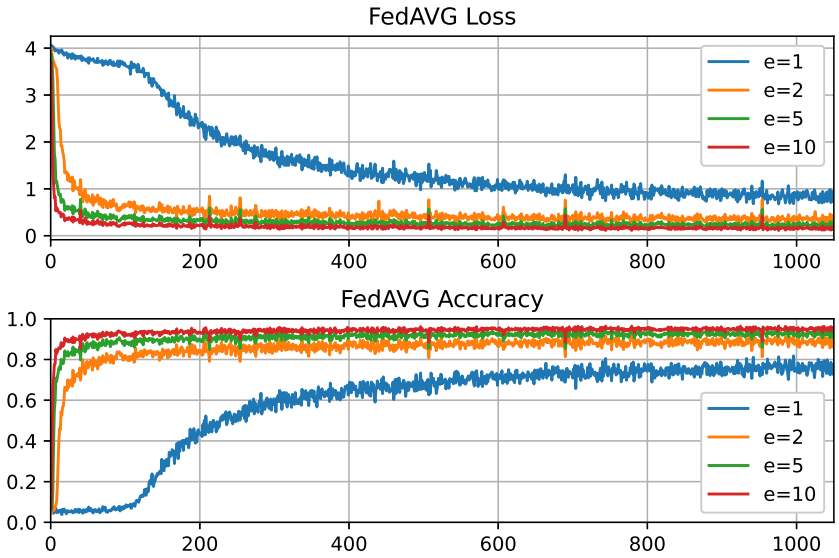

Figure 2. The impact of increasing local updated into Loss and Accuracy for Federated Average Algorithm

Considering only these findings, we can conclude that a high number of epochs yield better results. However, due to heterogeneity of data distribution, performing a high number of local updates could generate optimal local models. To investigate this line of thought, we evaluated the difference in collected metric values as well as the validation loss and accuracy. For instance, in training with only one epoch, the mean accuracy of the last 200 rounds was 75.28% and validation 77.13%. With 10 epochs the mean accuracy was 94.95% and the validation accuracy was 83.96%. The difference between metrics using epochs equals ten increased 5.94 times the gap between accuracy and validation

Table 1. Communication Rounds necessary to achieve target loss and accuracy

0.75 Loss	**Epoch 1**	**Epoch 2**	**Epoch 5**	**Epoch 10**
FedAVG	744	43	14	7
FedADAM	614	39	11	5
75% Accuracy	**Epoch 1**	**Epoch 2**	**Epoch 5**	**Epoch 10**
FedAVG	305	189	111	57
FedADAM	258	163	87	48

accuracy. In terms of loss, similar behavior was observed, the loss and validation loss for one and ten epochs were respectively: 0.8818 and 0.7565, and, 0.1613 and 0.6338. This comparison can be graphically analyzed in Figure 3.

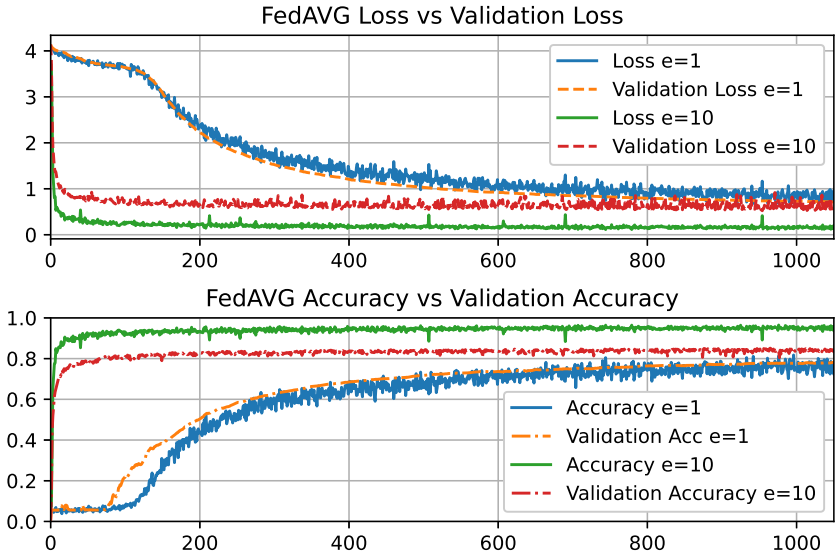

Figure 3. Comparative between loss/accuracy and validation loss/accuracy for epoch equals one and ten

We performed the same simulations using FedADAM and the increasing number of epochs also improved the results (Figure 4). The validation accuracy of ten epochs setting achieved the same result of one epoch. This demonstrates that increasing local updates for Adam optimizer can speed the convergence but no improved result was obtained at the end. Thus, the use of a high number of epochs for Adam optimizer is discouraged, since it will increase the time necessary to compute local updates.

We compared the Federated Average and Federated Adam algorithms as a function of the number of epochs. Results demonstrated that FedAVG produces better results for ten epochs, but when the number of communication rounds is not an issue, for instance in cross-silo settings, FedADAM provides faster convergence using one epoch per round. Considering the data collected through these simulations, we proposed an adaptation at the client-side, where the number of the local update should vary for less when using ten epochs settings. The goal is to diminish the overfitting and still increase the speed of convergence. This experiment is described in the next section.

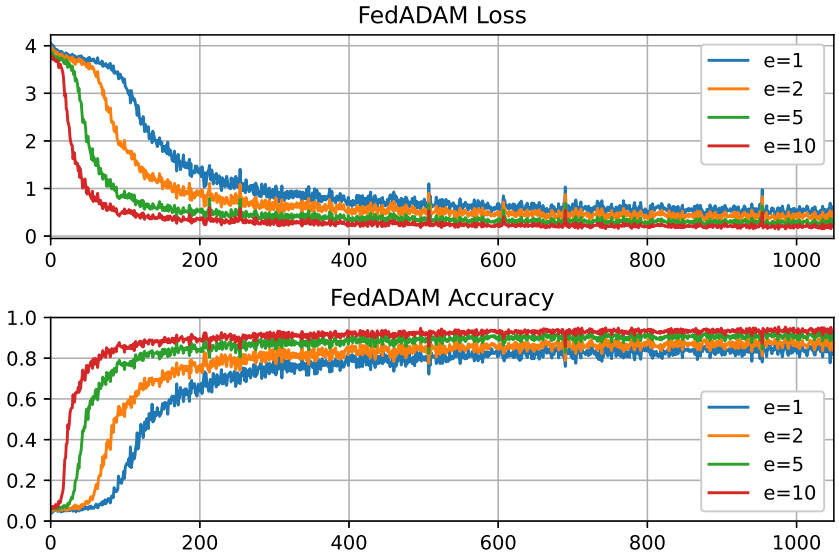

Figure 4. The impact of increasing local updated into Loss and Accuracy for Federated Adam Algorithm

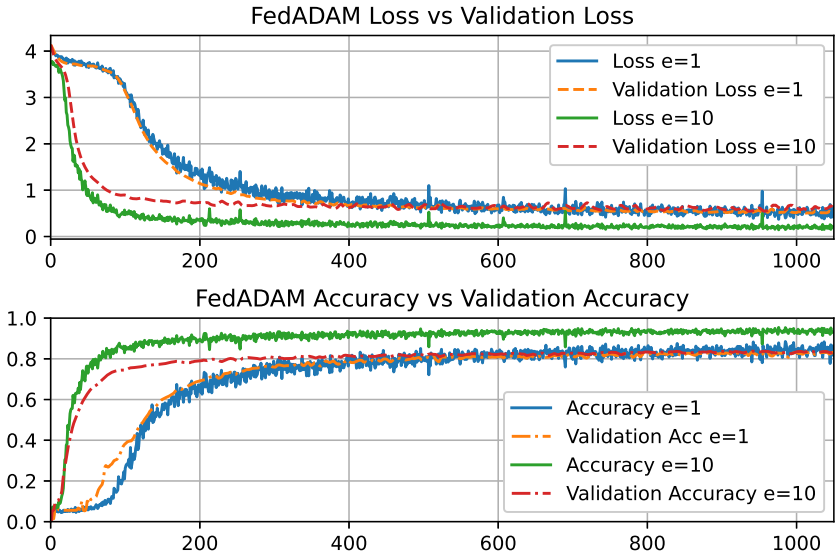

Figure 5. Comparative between accuracy and validation accuracy for epoch equals one and ten

5. Proposed adaptation and Results

In this section, we introduce a proposal for communication-efficient FL that tries to avoid communication overhead and overfitting due to the number of rounds pursued. The number of client epochs determines the amount of ”sequential progress” (or learning) each client makes before updating the global model. A high number of epochs results in more local progress each round, this can manifest as a much faster per-round convergence rate. The risk of overfitting may be correlated to non-IID distribution of client datasets. The less similar to the ”global” dataset each client dataset is, the more likely there will be ”drift” (clients converge to different optimal points) [Karimireddy et al. 2021] when using a high number of epochs during rounds closer to potential convergence.

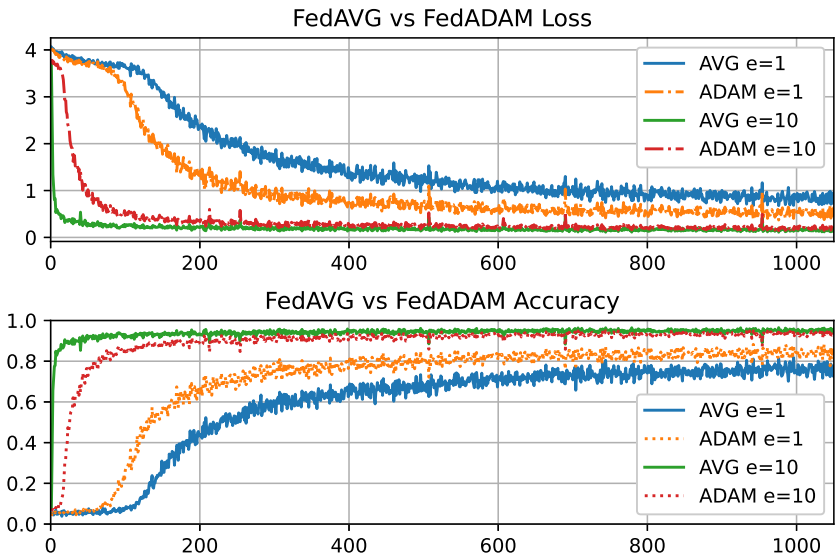

Figure 6. FedAVG works better than FedADAM for ten epochs

In this scenario, validation/test accuracy is less likely to improve. To reduces this negative effect, we added a controller at the client-side to vary the number of epochs and periodically reduces the local updates. We employed a Poisson distribution for that. We simulate two scenarios, the first with 1 epoch per round. The controller in this case periodically adds rounds using a Poisson distribution. In the second scenario, we considered 10 epochs per round, and the controller reduces the number of local updates, expecting to reduce the chance of yielding an optimum local. We performed 1500 rounds and set the probability of an event once in 1500 chances and for the other setting, we set ten times in 1500 rounds. Then, the limits of epochs were [1,10] for all settings.

Figure 7 shows the comparison of loss metrics for FedAVG before and after the adjustment for our first scenario, considering only one epoch as the default. As Poisson distribution will increase the number of local updates, but most of the time the probability will be concentrated around one epoch, the result improves just a little. It means that our approach doesn't boost enough the FedAVG. However, when considering the scenario with ten epochs that we analyze in the last section our method achieve considerable improvement. Figure 8 shows that using our variable number epochs, the gap between training and validation metrics is reduced. This behavior indicates that this model has less probability of overfitting. Even when it achieves less accuracy or higher loss, it still has better generalization and can be considered a better model according to these parameters.

The results of Figure 9 indicates that our approach can also be used with ADAM optimizer to improve model quality. Table 2 summarizes the results of both scenarios with FedAVG and FedADAM. For instance, the difference between loss and validation loss was 0.4541 before the adjustment and 0.0365 within. Also, the validation loss decreased 7.16% using the Poisson distribution controller. The same behavior happened with FedADAM, the difference between training and validation metrics reduces a lot and the comparison between validation accuracy with and without the adjustment showed a small decrease.

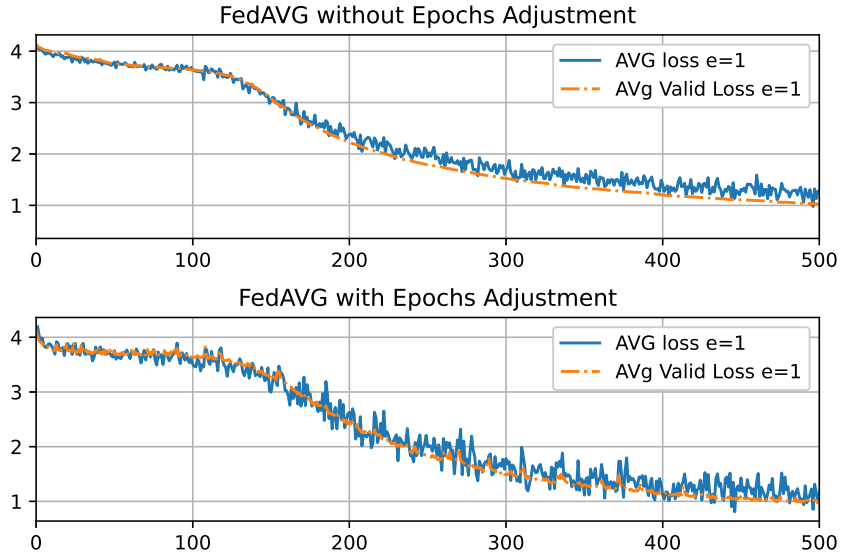

Figure 7. FedAVG Adjust using one epoch

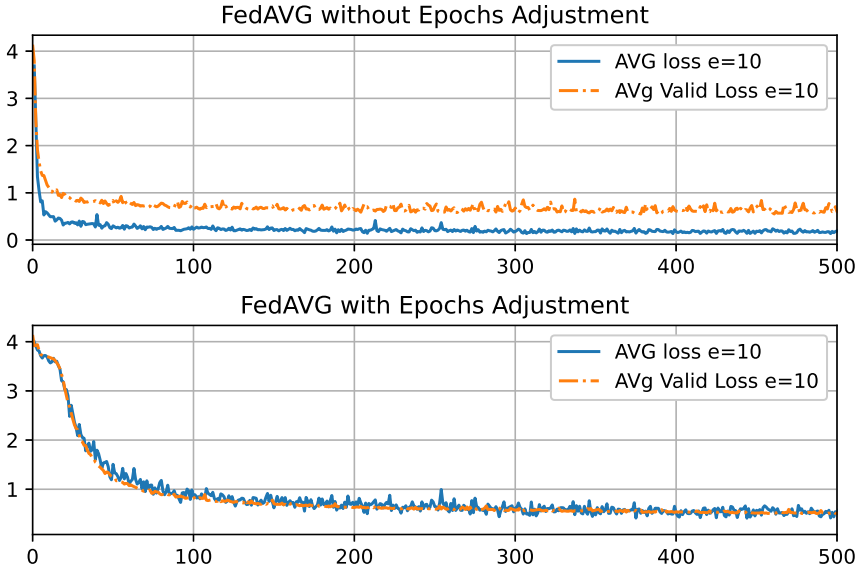

Figure 8. FedAVG Adjust using ten epoch

Table 2. Loss and Acc after our proposed adjustment

Metric between [200-300] rounds	Loss	Valid Loss	Acc	Valid Acc
FedAVG 10 Epochs	0.2046	0.6587	93.74%	82.61%
FedAVG 1 Epoch	1.9806	1.8252	52.61%	57.66%
FedAVG 10 E Adjust	**0.6480**	**0.6115**	**80.78%**	**80.14%**
FedADAM 10 Epochs	0.3175	0.6963	90.32%	79.86%
FedADAM 1 Epoch	1.0906	0.9330	71.11%	73.33%
FedADAM 10 E Adjust	**1.0962**	**0.9703**	**70.87%**	**72.43%**

6. Conclusion and Future Works

In this paper, we first performed an empirical evaluation of the impact of local updates on FedAVG and FedADAM performance. We showed that increase in the number of epochs

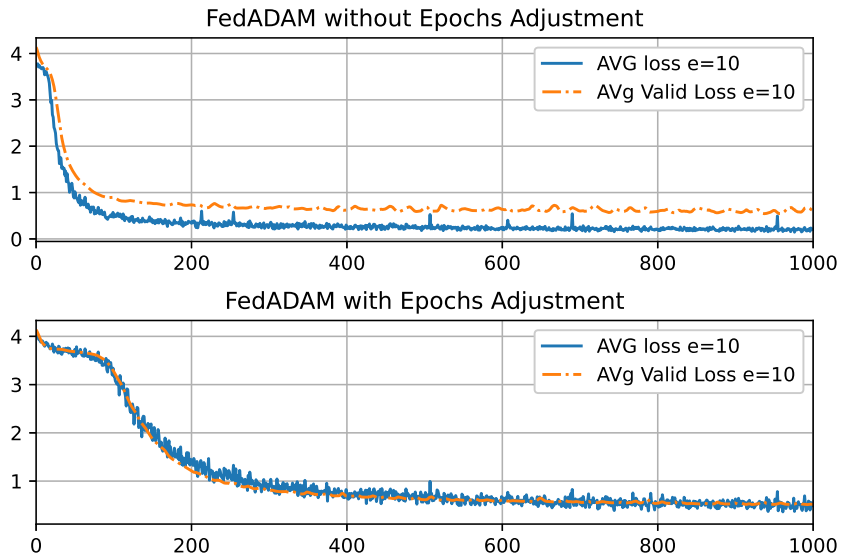

Figure 9. FedADAM Adjust using ten epoch

can speed up convergence but yields a high chance of producing overfitting (reducing the generalization). To mitigate the side effects of using too many local updates, we further propose the utilization of a controller that increases or reduces the number of epochs periodically to improve the model generalization. The probability of the number of epochs used was configured with Poisson Distribution, we verified the advantages of the proposed methods reducing the difference between training and validation metrics. We conclude that the number of epochs shouldn't be defined statically; it depends on other constraints such as connectivity, mobility, computing resource. For example, if the client has weak connectivity or the network is overload, it should dynamically increase the number of local updates, but eventually reducing it to avoid model problems.

As future works, we can extend the evaluation by adding more variables to the simulation as more sophisticated client selection criteria or evaluate how the drop-off of clients during training might impact on the overall performance. We might evaluate other tasks related to natural language processing (NLP) or regression to evaluate how our proposed method performs in different scenarios.

References

Abdulrahman, S., Tout, H., Ould-Slimane, H., Mourad, A., Talhi, C., and Guizani, M. (2021). A survey on federated learning: The journey from centralized to distributed on-site learning and beyond. *IEEE Internet of Things Journal*, 8(7):5476–5497.

Caldas, S., Duddu, S. M. K., Wu, P., Li, T., Konečný, J., McMahan, H. B., Smith, V., and Talwalkar, A. (2019). Leaf: A benchmark for federated settings.

Chen, M., Mathews, R., Ouyang, T., and Beaufays, F. (2019). Federated learning of out-of-vocabulary words.

Dutta, S., Joshi, G., Ghosh, S., Dube, P., and Nagpurkar, P. (2018). Slow and stale gradients can win the race: Error-runtime trade-offs in distributed sgd.

Hard, A., Rao, K., Mathews, R., Ramaswamy, S., Beaufays, F., Augenstein, S., Eichner, H., Kiddon, C., and Ramage, D. (2019). Federated learning for mobile keyboard prediction.

Hsu, T.-M. H., Qi, H., and Brown, M. (2019). Measuring the effects of non-identical data distribution for federated visual classification.

Kairouz, P., McMahan, H. B., Avent, B., and et al. (2021). Advances and open problems in federated learning.

Karimireddy, S. P., Kale, S., Mohri, M., Reddi, S. J., Stich, S. U., and Suresh, A. T. (2021). Scaffold: Stochastic controlled averaging for federated learning.

Li, T., Sahu, A. K., Zaheer, M., Sanjabi, M., Talwalkar, A., and Smith, V. (2020). Federated optimization in heterogeneous networks.

McMahan, H. B., Moore, E., Ramage, D., Hampson, S., and y Arcas, B. A. (2017). Communication-efficient learning of deep networks from decentralized data.

McMahan, H. B. and Streeter, M. (2010). Adaptive bound optimization for online convex optimization.

Melis, L., Song, C., Cristofaro, E. D., and Shmatikov, V. (2018). Exploiting unintended feature leakage in collaborative learning.

Mukherjee, A. and Rojas, B. (jul-2020). Business models for the long-term storage of internet of things use case data, idc report - market perspective,https://www.idc.com/getdoc.jsp?containerid=prap46737220.

Ramaswamy, S., Mathews, R., Rao, K., and Beaufays, F. (2019). Federated learning for emoji prediction in a mobile keyboard.

Reddi, S., Charles, Z., Zaheer, M., Garrett, Z., Rush, K., Konečný, J., Kumar, S., and McMahan, H. B. (2020). Adaptive federated optimization.

Reisizadeh, A., Mokhtari, A., Hassani, H., Jadbabaie, A., and Pedarsani, R. (2020). Fedpaq: A communication-efficient federated learning method with periodic averaging and quantization.

So, J., Guler, B., and Avestimehr, A. S. (2021). Turbo-aggregate: Breaking the quadratic aggregation barrier in secure federated learning.

Wahab, O. A., Mourad, A., Otrok, H., and Taleb, T. (2021). Federated machine learning: Survey, multi-level classification, desirable criteria and future directions in communication and networking systems. *IEEE Communications Surveys Tutorials*, 23(2):1342–1397.

Wang, H., Yurochkin, M., Sun, Y., Papailiopoulos, D., and Khazaeni, Y. (2020). Federated learning with matched averaging.

Wang, J., Xu, Z., Garrett, Z., Charles, Z., Liu, L., and Joshi, G. (2021). Local adaptivity in federated learning: Convergence and consistency.

Yu, T., Bagdasaryan, E., and Shmatikov, V. (2020). Salvaging federated learning by local adaptation.