Evaluating Methods to Automate Hyperparameter Tuning in Federated Learning

Rachel Phinnemore
Department of Computer Science
University of Toronto
rphinnemore@cs.toronto.edu

Yufei Kang
Department of Electrical Engineering
University of Toronto
yufei.kang@mail.utoronto.ca

Tianyu Wang
Department of Computer Science
University of Toronto
twang@cs.toronto.edu

***Abstract*—Edge computing is enabling the development of new ML frameworks, notably federated learning. However, there are challenges to optimize federated learning for edge devices due to the nature of the data on these devices (eg. distributed unequal, non iid data). In addition to these data challenges, edge computing devices are also resource constrained relative to the vast computation resources available on the cloud to train centralized ML models. In this paper, we investigate and experiment with ways to improve the model performance of federated learning specifically for edge computing devices. Specifically, we do this by marrying the success of FedAvg as the server aggregation algorithm with a comprehensive analysis of adaptive and non adaptive client optimizers. Thus, our experimentation directly addresses the unique constraints of optimizing federated learning for edge devices including the unique data distribution constraints as well as the constraints of limited on device training.**

I. Introduction

With forecasts that the number of IoT devices will grow exponentially this decade [1], edge computing will be increasingly permeating greater and greater folds of life. However, it is through empowering edge devices with machine learning that will enable us to fully leverage their capabilities. Yet, edge devices pose several unique challenges for machine learning including the data distribution and computation resources available locally on the devices. In light of this, a new type of machine learning has been developed that is particularly adapted to the configuration and requirements of edge devices, federated learning. Federated learning is a decentralized form of machine learning whereby a model is trained locally on a client's devices whose model weight results are then aggregated to form the global model [2]. Federated learning is seen as the ideal operating system for edge devices as it provides both a learning protocol for coordination as well as privacy [3]. This is important because edge devices are increasingly privy to private information as well as require compliance with stringent privacy regulations such as GDPR [4]. While federated learning is well adapted to edge devices, there are several open challenges regarding how to optimize its performance in light of constraints such as the limited opportunity for model hyperparameter tuning.

Hyperparameter tuning is one method of significantly improving the performance of ML systems [5]. Yet the opportunity for hyperparameter tuning in federated learning may be constrained or impossible due to the limited number of communication rounds [6]. As such, alternative methods must be deployed to optimize the performance of federated learning models in lieu of hyperparameter tuning. Specifically, one method of improving federated learning model performance without hyperparameter tuning involves using schedulers that will automatically or adaptively adjust the learning rate throughout training [6].

In this paper, we present a foundational evaluation of methods to automate hyperparameter tuning (eg. client or server learning rates) of federated learning on the CNN model using the FMNIST and MNIST datasets. We conduct this evaluation through three experiments that automatically decay or adaptively adjust the learning rate. Firstly, in Experiment 1, we implement FedAvg as the server optimizer and a non-adaptive client optimizer, SGD configured with three methods to automatically decay the learning rate (eg. StepLR, PlateauLR, CosineLR). In Experiment 2, we implement FedAvg as the server optimizer with three adaptive client optimizers (eg. Adam, Adadelta, Adagrad). Finally, in Experiment 3, we implement a server learning rate in addition to a client learning rate. In this experiment, we implement FedAvg as the server optimizer paired with three methods to automatically decay the server learning rate with SGD as the client optimizer paired with plateau LR to decay the client learning rate. To the best of our knowledge, this paper is the first to evaluate the automation of learning rate tuning using distinct learning rate decay schedulers for the client and server learning rates in federated learning, as conducted in Experiment 3.

II. Related Works

There are several active lines of research seeking to improve various aspects of the performance of federated learning for heterogeneous data found in edge computing. The works that are most closely related to ours are Felbab [7], McMahan [8], and Charles [6]. Firstly, our paper was inspired by the work of Felbab [7] who sought to optimize federated learning through different client optimizers. Specifically, in their work, they implement the FedSGD server optimizer with various client optimizers including SGD, Adam, Adagrad, Nesterov momentum, and RMSProp for NIID balanced data on the FMNIST dataset. However, since the work by Felbab [7], a new server optimizer, FedAvg [8] has gained prominence. Thus, in Experiment 1 and 2, we adopt a similar approach as Felbab [7] of experimenting with various client optimizer schemes, while using the new FedAvg as the server optimizer. FedAvg was presented by McMahan [8] as a federated learning model that

includes iterative model averaging. It was able to achieve vast success in reducing the number of communication rounds required by a factor of 10 - 100x [8]. In the FedAvg implementation, they use FedAvg as the server optimizer and SGD as the client optimizer and run experiments on five different model architectures and four datasets. In terms of engineering implementation, our work is most closely related to the work of McMahan [8] as we have built our implementation on top of their codebase available to the public on Github [9]. However, in the work by McMahan [8] they tie the client and server learning rate together as noted by Charles [6]. Further, Charles [6] illustrate that optimizing a model for best performance requires decoupling the client and server learning rate. Thus, we differentiate ourselves from McMahan by exploring the possibility of realizing greater model performance by implementing a separate client and server learning rate in Experiment 3. Finally, our work is closely related to the work by Charles [6] who present a method of automatically tuning the client and server learning rate through learning rate decay to mitigate the need to manually tune these hyperparameters. However, we differentiate our work by exploring additional ways to automate the client learning rate in Experiment 1 as well as adaptively adjusting the client learning rate in Experiment 2. Finally, in the work by Charles [6], they automatically tune the client and server learning rate using an implementation that employs SGD as both the client and server optimizer. In contrast, in Experiment 3, we implement an adaptive client optimizer paired with FedAvg on the server-side alongside 3 strategies to automatically decay the server learning rate.

III. Implementation

A. Overview of FedAvg

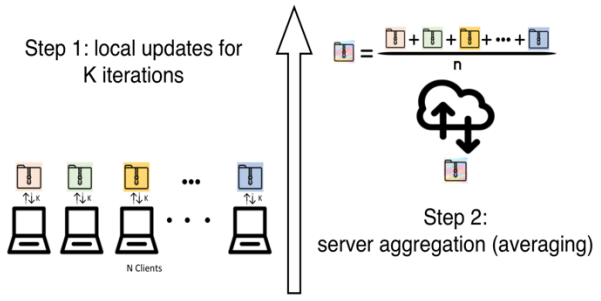

Fig 1: Visualization of FedAvg

In this work, we look into the most popular federated learning algorithm, Federated Averaging [8] as the server optimizer. In FedAvg, local data on remote clients are assumed to be independent and identically distributed. At each communication round of FedAvg, the central parameter server first selects a small set of clients and then broadcasts the latest global model to them. After receiving the latest model, selected clients perform model training locally on their private datasets and send the model updates back to the central server later. Afterward, the central server aggregates all the local updates from clients by performing averaging and synchronizes the global model. In this way, FedAvg trains machine learning models without requiring clients' raw data and therefore offers privacy protection. Meanwhile, demonstrated by empirical experiments, models trained by FedAvg achieve good accuracy comparable to that trained in the central setting when the i.i.d. assumption is satisfied.

B. Experiment Setup

- **Datasets:** To compare different schedulers and optimizers, we train the CNN model with different schedulers and optimizers on MNIST and FMNIST dataset. The MNIST dataset contains 70000 examples of grayscale handwritten digits from 10 classes, the FMNIST dataset contains 70000 examples grayscale images of fashion items from 10 classes, both with a train/test split.
- **Models:** For MNIST, we train a CNN model with two 5 times 5 convolution layers. The first layer has 10 output channels and the second has 20, followed by a dropout layer. Then, two fully connected layers with 50 output channels and 10 output channels respectively, are implemented. For FMNIST dataset, we train a CNN model with two 5 times 5 convolution layers. The first layer has 16 output channels and the second has 32, with each followed by a batch norm layer and a max pooling layer. Then, a fully connected layer with 10 output channels is implemented. We used both models as per their implementation by McMahan [8].
- **Training:** The models are trained for 30 communication rounds, with 100 clients, each client will perform 10 epochs of local training for each communication round. We selected 30 communication rounds as this number of communication rounds was used by Felbab [7] and worked with our constraint of limited computation resources. The number of clients used as well as the number of clients per round was left at the default set by the work of McMahan [8].
- **Data Distribution:** The dataset is split across the clients with NIID distribution but an equal number of samples as was done in Felbab [7]. Specifically, NIID equal setting refers to each client having an equal number of data points of varying distributions (eg. client 1:[4,1,9], client 2:[2, 8, 7]). Further, the reason for focusing on equal NIID data is that we are focused on improving the performance for the consumer edge devices such as smartphones, and smart home devices which are expected to have a fairly uniform number of data samples across devices.
- **Hyperparameters:** The default server learning rate is set to 1 as per implementation in Charles [6] and the default client learning rate is set to 0.01 as per the implementation in McMahan [8] unless the optimizer has a default learning rate, other hyperparameters for each scheduler and optimizers are also set to default.
- **Experiment Metrics:** In terms of experiment metrics, we recorded the accuracy and loss at each communication round to compare the performance of different methods. We also record run time, although we do not offer an analysis on runtime due it's bias from the specific computer used to run the experiments.

- **Experiment Mechanisms:** As we were unable to find prior work implementing either learning schedulers or multiple adaptive client optimizers for the FedAvg, we arbitrarily selected the learning rate schedulers and adaptive optimizers based on their popularity and prior success in the ML community for Experiment 1 and 2. For Experiment 3, we choose to use the Plateau LR learning rate scheduler with the client optimizers given that it had the best performance overall in Experiment 1 as well as prior experimentation. In Experiment 3, we also choose to use the Adam optimizer as the adaptive optimizer due to it's popularity in the machine learning community.

C. *Experiment 1 – Client Learning Rate Schedulers*

Motivation: In the face of scarce communication rounds for hyperparameter tuning in federated learning, Charles [6] illustrated a method for automatically decaying the client and server learning rate together as opposed to tuning the learning rates to great success. However, the performance of decaying client and server learning rate as a unit varies on different ML tasks and data distributions. As such, it is a difficult and delicate task to find such balance for each ML task and data distribution. Therefore, we are motivated to separate the client learning rate adjustment and evaluate how the client learning rate itself influences the performance of federated learning. In Experiment 1, we simplify this approach by exploring three methods to automatically tune the client learning rate exclusively through three schedulers for the SGD local solver in Experiment 1, namely, StepLR, PlateauLR and CosineLR.

Method: By implementing these schedulers on the client-side, the local solver is allowed to dynamically reduce the learning rate based on some validation measurements.

StepLR scheduler: With pre-fixed step size and decay unit, StepLR allows to decay the learning rate of each parameter group by a decay unit every step_size epochs.

PlateauLR scheduler: Via monitoring a metrics quantity, PlateauLR scheduler reduces learning rate when a metric has stopped improving. Notably, this scheduler would wait for some extra number of epochs to make sure learning indeed stagnates.

CosineLR scheduler: CosineLR scheduler sets the learning rate of each parameter group according to cyclical learning rate policy (CLR). The policy cycles the learning rate between two boundaries with a constant frequency. Notably, the distance between the two boundaries can be scaled on a per-iteration or per-cycle basis.

Results: As can be seen in Table 1, CosineLR optimizer had the highest accuracy for the MNIST dataset while the PlateauLR optimizer had the highest accuracy for FMINST. This illustrates that the client learning rate may need to be selected according to the dataset and task on hand. Ultimately, the FedAvg + SGD + Plateau LR result was our most significant result as it was the only result to outperform the vanilla FedAvg implementation from Experiment 2, achieving 94.83% accuracy compared to 94.70%. One reason that Plateau LR may have outperformed vanilla FedAvg is PlateauLR scheduler reduces learning rate when a metric has stopped improving. In this way, learning rate decays more cautiously than other schedulers.

TABLE I. EXPERIMENT 1 RESULTS

Dataset	**Experiment 1 Results**		
	FL Model Configuration	*Test Accuracy*	*Runtime*
MNIST	FedAvg + SGD + StepLR	88.24%	1035.82s
MNIST	FedAvg + SGD + PlateauLR	88.13%	2549.55s
MNIST	**FedAvg + SGD + CosineLR**	**92.76%**	**2831.43s**
FMNIST	FedAvg + SGD + StepLR	91.85%	2120.20s
FMNIST	**FedAvg + SGD + PlateauLR**	**94.83%**	**5949.94s**
FMNIST	FedAvg + SGD + CosineLR	84.78%	33652.85

D. *Experiment 2 – Adaptive Client Optimizers*

Motivation: While SGD has proven successful as a client optimizer in several implementations, there are promising developments in the realm of adaptive optimizers as well. One notable instance is the work by Xie [10] where they present a novel SGD client optimizer variant modelled on the adaptive optimizer, Adagrad. However, while they evaluate the performance of Adagrad relative to the performance of their proposed Adagrad and find that their solution drastically reduces communication rounds required for convergence, they fail to evaluate the performance of existing adaptive client optimizers relative to non adaptive optimizers such as SGD. Thus, we were inspired to evaluate how adaptive optimizers paired with FedAvg previously not evaluated by the research community such as Adam, Adagrad, Adadelta perform relative to non adaptive optimizers such as SGD.

Method: As such, we introduce advanced adaptive optimizers in the local update step to replace the vanilla SGD optimizer (with learning rate schedulers). With adaptive optimizers, clients are enabled to train the local models adaptively based on the learning process rather than following a manually designed schedule. In this work, we implement three popular adaptive optimizers which have already shown great success in the machine learning community.

Adam optimizer: Adaptively estimating the lower-order moments, Adam optimizer solves the first-order gradient-based optimization of stochastic objective functions efficiently. Moreover, this method is also appropriate for non-stationary objectives and problems with noisy and/or sparse gradients.

Adagrad optimizer: Adagrad is a subgradient method that dynamically incorporates knowledge of the geometry of the data observed in earlier iterations and further performs more informative learning based on that. Metaphorically, the adaptation helps find needles in haystacks in the form of very predictive but rarely seen features.

Adadelta optimizer: Adadelta is a per-dimension learning rate method for gradient descent. In particular, the method dynamically adapts over time using only first-order information and has minimal computational overhead beyond vanilla stochastic gradient descent. Notably, Adadelta requires no manual tuning of a learning rate and appears robust to noisy gradient information, different model architecture choices, various data modalities, and selection of hyperparameters.

Results: As illustrated in Table 2, across both datasets of Experiment 2, FedAvg + SGD had the best performance. Additionally, FedAvg had the best overall performance for the MNIST dataset from all three experiments as well as provided the second strongest performance on the FMNIST dataset across all three experiments. This shows the robustness of the FedAvg + SGD algorithm. Unfortunately, the FedAvg implementations with the adaptive optimizers, Adam, Adadelta, and Adagrad did not achieve competitive accuracy nor run times. One possible reason for the adaptive optimizer's poor performance is that our experiments are based on the gradient. We tried to use gradients to adjust the whole system but this may not be enough. We may need more information and parameters to properly automatically tune (eg. past gradients, local data distribution,). We are adjusting based on current gradient instead of past gradients. (learning trajectory could be utilized to predict future). Also, the server has no idea about the local distribution, and clients have no idea about others' data distribution. They can only perform following optimizers W/O schedulers rather than taking empirical conditions into consideration.

TABLE II. EXPERIMENT 2 RESULTS

Dataset	Experiment 2 Results		
	FL Model Configuration	*Test Accuracy*	*Runtime*
MNIST	**FedAvg + SGD**	**93.16%**	**1021.14s**
MNIST	FedAvg + Adam	86.23%	1103.74s
MNIST	FedAvg + Adadelta	87.63%	1119.67s
MNIST	FedAvg + Adagrad	78.50%	1068.44s
FMNIST	**FedAvg + SGD**	**94.70%**	**2184.44s**
FMNIST	FedAvg + Adam	88.53%	2373.64s
FMNIST	FedAvg + Adadelta	93.42%	2509.33s
FMNIST	FedAvg + Adagrad	39.02%	2339.49s

E. Experiment 3 – Server Learning Rate Schedulers

Motivation: Having conducted an extensive evaluation of the performance of various client optimizers in Federated Learning, we proceed to investigate the role of decoupling the client and server learning rate on model performance. This was motivated by the work from Charles [6] who found that there is an opportunity to improve performance through mechanisms that decouple the global model and local updates, and introduce the concept of server learning rate. As such, we decouple the server learning rate using the mechanism outlined below, implement three different server learning rate decay mechanisms with the FedAvg server optimizer and evaluate their performance while using both an adaptive (eg. Adam optimizer) and non adaptive client optimizer (eg. SGD with the plateau LR scheduler).

Method: In FedAvg, a vanilla averaging is applied when computing the new global model. Before introducing our improvement, we first rewrite the server averaging formula as below.

$$x_{t+1} = \frac{1}{|\mathcal{S}|}\sum_{i\in\mathcal{S}} x_i^t = x_t - \frac{1}{|\mathcal{S}|}\sum_{i\in\mathcal{S}}\left(x_t - x_i^t\right)$$

Fig 2: Server Aggregation Formula

In this way, we can view the server aggregation step as a SGD optimization step by considering the delta as a pseudo-gradient on the server side. Then, similar to what we did before, we wish to make the server aggregation more efficient by improving the basic SGD optimization method. Therefore, we introduce learning rate decay on the server-side, and three global learning rate schedulers we implemented in this work:

Loss-based scheduler: By monitoring the average loss, the loss-based scheduler decays the server learning rate by decay rate when the loss has not decreased for some iterations. In this way, the variance of loss is used as feedback of the server scheduler.

Communication round-based scheduler: In this method, a counter is added to count the number of communication rounds of the current federated learning task. Based on that number, the scheduler decays the server learning rate.

Exponential decay scheduler: With pre-fixed maximal and minimal learning rate as well as decay rate, exponential decay scheduler divides the server learning rate by the decay rate gradually until the minimal learning rate is reached.

Results: In Experiment 3, we implement an adaptive client optimizer, Adam and a non client optimizer, SGD with the Plateau LR learning rate scheduler with the FedAvg server optimizer paired with three server learning rate schedulers. Firstly, the results showed that all configurations with the adaptive client optimizer, Adam underperformed relative the configuration with the non adaptive SGD optimizer. This is not surprising and follows the results of Experiment 2. Secondly, in terms of the three server learning rate schedulers - decay via loss, decay via communication round and exponential decay, in each experiment setting for both datasets, the exponential decay resulted in the highest performance. Thus, the robustness of the exponential decay as a server learning rate scheduler is a significant finding from this experiment. We hypothesize that the success of the exponential decay server learning rate scheduler is due to its performance as it approaches the final training rounds. During the final training rounds, the exponential decay guarantees that the learning rate will not be decayed to a small scale number. This enables the optimizer to continue to learn during these final rounds and thus provides a path to the best optimization. In contrast, the other learning rate schedulers will approach a small scale number which can cause the optimization to plateau (eg. cease learning and improving performance). Finally, Fig. 3 and Fig. 4 illustrate two unique strengths of the exponential decay server learning rate scheduler. Firstly, between communication round 0 to 10, the exponential scheduler enables rapid loss decay, which is an advantageous property during early communication rounds. Secondly, these graphs show how the exponential decay server learning rate scheduler converges to a lower loss stably during the final communication rounds while the other learning rate schedulers show greater variability. This can be seen by comparing the loss level at communication round 25 and round 30.

TABLE III. EXPERIMENT 3 RESULTS

Dataset	Experiment 3 Results		
	FL Model Configuration	*Test Accuracy*	*Runtime*
MNIST	FedAvg + Server LR Decay via Loss + Adam client optimizer	77.36%	3306.78s
MNIST	FedAvg + Server LR Decay via Com. Round + Adam client optimizer	70.64%	2593.13s
MNIST	**FedAvg + Server LR Decay via Exp. Decay + Adam client optimizer**	**78.37%**	**1332.27s**
MNIST	FedAvg + Server LR Decay via Loss + SGD + PlateauLR	88.45%	2441.53s
MNIST	FedAvg + Server LR Decay via Com. Round + SGD + PlateauLR	89.60%	2458.11s
MNIST	**FedAvg + Server LR Decay via Exp. Decay + SGD + PlateauLR**	**92.88%**	**989.59s**
FMNIST	FedAvg + Server LR Decay via Loss + Adam	82.50%	7692.54s
FMNIST	FedAvg + Server LR Decay via Com. Round + Adam	72.34%	9936.82s
FMNIST	**FedAvg + Server LR Decay via Exp. Decay + Adam**	**88.31%**	**2326.54s**
FMNIST	FedAvg + Server LR Decay via Loss + SGD + PlateauLR	87.32%	40146.45s
FMNIST	FedAvg + Server LR Decay via Com. Round + SGD + PlateauLR	93.11%	7991.28s
FMNIST	**FedAvg + Server LR Decay via Exp. Decay + SGD + PlateauLR**	**93.27%**	**2149.755s**

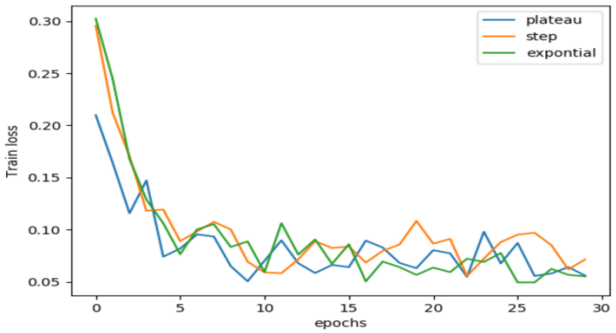

Fig 3: Experiment 3 MNIST dataset with 3 server learning rate decay schedulers + SGD client optimizer + Plateau LR client learning rate scheduler

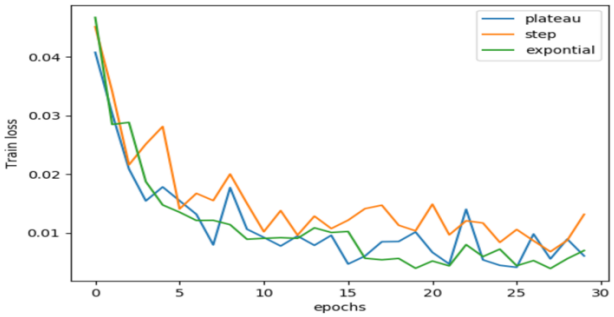

Fig 4: Experiment 3 FMNIST dataset with 3 server learning rate decay schedulers + SGD client optimizer + Plateau LR client learning rate scheduler

CONCLUSION

We have provided an extensive evaluation of mechanisms to automate hyperparameter tuning for a prominent machine learning paradigm for edge devices, namely, Federated Learning. In doing so, we have contributed several findings to the field. Notably, the success of the Plateau LR client learning rate scheduler on the FMNIST dataset in Experiment 1 which slightly outperformed the robust "vanilla" FedAvg and SGD configuration as seen by comparing to results from Experiment 2. In Experiment 2, we find that existing adaptive client optimizers fail to achieve significant performance. However, there is a promising approach in adapting the theory of these optimizers to Federated Learning as shown by Xie [10]. Finally, in Experiment 3, we implement a server learning rate and demonstrate the strength of the exponential decay server learning rate scheduler across both datasets. To conclude, as the number of edge devices proliferates, it is important to continue to expand the possibility of richer functionality on these devices via machine learning.

FUTURE WORKS

Firstly, while it is important to develop mechanisms to improve federated learning, a key component is testing these mechanisms on devices in the wild. Thus, the most critical area to extend our work is deploying the methods we present and evaluating them on devices. Beyond this step, it is also important to test the configurations that we present on NIID unequal data as our assumption that data on consumer devices will mimic NIID equal data may not hold. Finally, it is important to explore client and server learning rate mechanisms beyond those that we propose, as well as testing the mechanisms we propose on additional datasets and models to determine generalizability.

REFERENCES

[1] *Cisco Visual Networking Index: Global Mobile Data Traffic ...* www.ramonmillan.com/documentos/bibliografia/VisualNetworkingIndexGlobalMobileDataTrafficForecastUpdate2016_Cisco.pdf.

[2] Jere, Shashank et al. "Federated Learning in Mobile Edge Computing: An Edge-Learning Perspective for Beyond 5G." *ArXiv* abs/2007.08030 (2020): n. pag.

[3] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. 2019. Federated Machine Learning: Concept and Applications. ACM Trans. Intell. Syst. Technol. 10, 2, Article 12 (February 2019), 19 pages. DOI:https://doi-org.myaccess.library.utoronto.ca/10.1145/3298981

[4] Li, Li, et al. "A Review of Applications in Federated Learning." *Computers & Industrial Engineering*, Pergamon, 18 Sept. 2020, www.sciencedirect.com/science/article/pii/S0360835220305532.

[5] Wu, Jia, et al. "Hyperparameter Optimization for Machine Learning Models Based on Bayesian Optimization." *Journal of Electronic Science and Technology*, Elsevier, 11 Dec. 2019, www.sciencedirect.com/science/article/pii/S1674862X19300047.

[6] Charles, Zachary, and Jakub Konečný. "On the Outsized Importance of Learning Rates in Local Update Methods." *ArXiv.org*, 2 July 2020, arxiv.org/abs/2007.00878.

[7] Felbab, Vukasin, et al. *Optimization in Federated Learning*. CEUR-WS, 2019, ceur-ws.org/Vol-2473/paper13.pdf.

[8] McMahan, H. B. et al. "Communication-Efficient Learning of Deep Networks from Decentralized Data." *AISTATS* (2017).

[9] RJ, Ashwin. "AshwinRJ/Federated-Learning-PyTorch." *GitHub*, github.com/AshwinRJ/Federated-Learning-PyTorch.

[10] Xie, Cong & Koyejo, Oluwasanmi & Gupta, Indranil & Lin, Haibin. (2019). Local AdaAlter: Communication-Efficient Stochastic Gradient Descent with Adaptive Learning Rates.