Federated Learning: Ablation Study and Adaptive Optimization

Francesco Cartelli
cartelli.francesco@gmail.com

Francesco Torta
francesco.torta98@gmail.com

Giorgio Voto
giorgiovoto@gmail.com

Computer Engineering Department,
Politecnico di Torino

Abstract

Federated learning is a distributed machine learning paradigm in which many clients collaborate under the orchestration of a central server, in order to train an algorithm without sharing their own training data. While typical distributed models in data center are trained using data independent and identically distributed, clients data are typically far from IID. In this paper, standard federated methods such as Federated Averaging (FedAVG), as well as FedIR and FedVC are firstly tested to verify their improvements in accuracy and stability for a visual classification task. Adaptive optimization is secondly applied to increase the performance of the standard algorithms and is addressed by various optimizers: Adagrad, Adam and SGD. Tests are focused on verifying the convergence and stability of the new model in presence of heterogeneous data. Final results highlight that the use of adaptive optimizers significantly improve the performance of the model.

1. Introduction

Federated Learning (FL) is a machine learning paradigm which embodies the preservation of data privacy: training model and data are kept decentralized to mitigate the confidentiality loss risk. In this scenario the challenges concern the unreliability of the network connection and, simultaneously, the large heterogeneity of in the data allocated to each client.

The most popular algorithm for this setting is **FedAVG** (Federated Averaging) [6]. In this method, the model needs to be iteratively trained over several rounds. In every round, each active client receives an initial model from a central server, performs stochastic gradient descent (SGD) with a local data batch and sends back the updated model parameters. It is the task of the server to provide data aggregation for the received response and the subsequent update of the central model. In the paper [2] two different variations are proposed:

- **FedIR**, which applies *Importance Reweighting* for learning on datasets distributed differently from a target distribution.
- **FedVC**, used to overcome difficulties in processing clients with overly large number of samples.

A variation of this popular algorithm is **FedOPT** (Federated Optimization), which incorporate knowledge of past iterations to perform more informed optimization [7]. In this paper the technique is incorporated and expanded by applying new optimizers.

In this work, we primarily focus on comparing and study different configuration of the standard federated methods (FedAVG, FedIR and FedVC) in respect to a canonical centralized network. We focus in particular on *Non-IID Class Distribution*. The diversity of each client class distribution is addressed by parametrically change the concentration parameter (α) of a *Dirichlet distribution*. by producing client with a variable number of samples and also different classes distributions.

2. Related Works

Federated Visual Classification with Real-World Data Distribution. [2] In this paper, the effect that real-world data distributions have on distributed learning are studied as a benchmark for the standard Federated Averaging (FedAvg) algorithm. To do so, two large-scale datasets for species and landmark classification were used. For the task, two new algorithms (FedVC, FedIR) were developed. These new methods intelligently re-sample and re-weight over the client pool, bringing large improvements in accuracy and stability in training.

Adaptive Federated Optimization. [7] The paper proposes a federated version of adaptive optimizers, including Adagrad, Adam, and Yogi. Various experiments prove their convergence in the presence of heterogeneous data. The extensive experiments on these methods show that the use of adaptive optimizers can significantly improve the performance of federated learning.

3. Methods

In the following, we describe the baseline approach of Federated Learning algorithm. Federated Averaging (FedAVG) is proposed with two variations: FedIR and FedVC, specifically addressed to overcome heterogeneous client distributions scenarios. We also try to change the network introducing batch and group normalization layers. The standard approach of FedAVG is then modified by the Federated Adaptive Optimization. Various optimizers are adopted by client and server, such as: Adam, Adagrad and SGD.

3.1. Federated Averaging (FedAVG)

The standard algorithm proposed [2], is the starting point for this work. In the process, which involves K clients and one server, the training is divided into N rounds. At each round, the central node of the network, delivers its model parameters to a subset of randomly selected devices. Each selected client receives the parameters from the server and performs local SGD optimization over its local mini batch. Client computations involve a specified number of local epochs before sending back the newer optimized model parameters. Each set of parameters received by the server is accumulated from the reporting client. A weighted average, based on each client samples number, is computed by the central node. The resulting average is applied to the previous model parameters to compute a new updated model. From this initial process multiple variations can be considered.

Importance Reweighting (FedIR) is an improvement over FedAVG aimed to increase accuracy for *non-identical class distribution* shift in federated clients. In the method, the objective function is to minimize the expected loss in respect to the target distribution $E_p[l(x, y)]$ (where x represent the images and y the labels). Since each client training data are sampled for a specific local distribution, each client loss function is a *biased* estimator of the loss with respect to the target distribution. For each client k, with distribution $q_k(x, y)$, we have $E_{q_k}[l(x, y)] \neq E_p[l(x, y)]$.

The reweighting scheme proposes the introduction of a weight $w_k(x, y)$ computed as follow

$$w_k(x, y) = \frac{p(x, y)}{q_k(x, y)} = \frac{p(y)p(x|x)}{q_k(y)q_k(x|y)}$$

This result can be computed directly on the client training example distribution: $q_k(y)$ is a private information that never leaves the client, $p(y)$ is transmitted by the server with minimal communication cost.

Virtual Clients (FedVC) proposes a redistribution algorithm for reducing disparity in imbalanced clients dataset size. A client with an overly large training dataset will take a long time to compute updates and takes significantly more local optimization steps than another with fewer training examples, creating a bottleneck in the federated learning round.
The algorithm aims to reduce the size of large clients, by splitting their local datasets into smaller ones. Each one of those new datasets is a *virtual* client.
At each round, each client selects a subset of N_{VC} number of samples. Given n_k as the size of D_k dataset, N_{VC} is a fixed parameter such that $N_{VC} \leq n_k$, when sampling is performed without replacement or $N_{VC} > n_k$ otherwise. To reduce the possibility of having underutilized samples from the larger clients, the probability of selecting one device is proportional to the size of its local dataset.

3.2. Normalization Layers

In this section, we try to introduce Normalization layers after each convolutional layer. This technique is typically used to reduce *internal covariate shift* and allow higher learning rates, more regularization and less dependency on initialization [3]. A normalizing transformation $\text{BN}_{\gamma,\beta} : x_{1...m} \rightarrow y_{1...m}$ is applied to every mini-batch to learn parameters γ and β, where

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}}$$

is the normalization and

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \text{BN}_{\gamma,\beta}(x_i)$$

allows to scale and shift normalized values to recover the identity mapping.

In a federated scenario, clients have datasets with different *class distribution non-IID*, but can also have feature shift (also known as *feature shift non-IID*) [5]. Local normalization can be useful in these contexts to alleviate the feature shift before the averaging step.

The learned parameters are sent from each client to the server, which will compute the weighted averages.

3.3. Adaptive Federated Optimization

The FedAVG algorithm suffers from unstable convergence when applied to clients with extremely heterogeneous distributions (e.g. Dirichlet distribution with $\alpha <= 1$), performing multiple epochs of SGD on their datasets.

This often leads to considerable instability as regards the convergence of the model, since the update are performed in a non-adaptive way. Adaptive Federated Optimization [7] tries to partially solve this problem introducing a gradient-based server optimizer, instead of averaging client updates. Suppose that at round t, the server has model x_t and samples a set $\mathcal{K}$ of clients, each containing n_k samples. Let x_k^t denote the model of each client $k \in \mathcal{K}$ after local training. Considering

$$\Delta_t = \sum_{k \in \mathcal{K}} \frac{n_k}{n_{tot}} (x_i^t - x_t)$$

We can rewrite FedAVG update as

$$x_{t+1} = x_t + \Delta_t$$

As we can see, FedAVG is equivalent to applying SGD to $-\Delta_t$ with learning rate $\eta = 1$.

Algorithm 1 FedOPT: Federated Optimization

Input: x_0, $ClientOPT$, $ServerOPT$
for $t = 0, \cdots, T-1$ **do**
  Sample a subset $\mathcal{K}$ of clients
  $x_{i,0}^t = x_t$
  for each client $i \in \mathcal{K}$ **in parallel do**
    for $e = 0, \cdots, E$ **do**
      Several $g_{i,e}^t$ of $\nabla F_i(x_{i,e}^t)$
      $x_{i,e+1}^t = ClientOPT(x_{i,e}^t, g_{i,e}^t, \eta_l, t)$
    end for
    $\Delta_i^t = x_{i,E}^t - x_t$
  end for
  $\Delta_t = \sum_{n_k \in \mathcal{K}} \frac{n_k}{n_{tot}} \Delta_i^t$
  $x_{t+1} = ServerOPT(x_t, -\Delta_t, \eta, t)$

The family of algorithms, called FedOPT, formalized in Algorithm 1 proposes to generalize the optimizer used in the update of the global model. $ServerOPT$ is the gradient-based optimizer that can be implemented with well-known optimizers as SGD, Adagrad or Adam, although they have not been designed for the federated scenario.

With this feature, the update does not take into account only the last round, but also the previous knowledge, thus increasing stability and reducing the non-convergence phenomena.

SGD (2) is an optimizer that allows the use of a learning rate (possibly adaptive), and a momentum that regulates the variation of the model parameters based also on the previous updates knowledge.

Algorithm 2 SGD

Input: γ(lr), x_0(params), $F(x)$ (objective), λ (weight decay), μ (momentum), τ (dampening)
$g_t = \nabla_x F_t(x_{t-1})$
computing weight decay...
if $t > 1$ **then**
  $g_t = \mu g_{t-1} + (1 - \tau) g_t$
end if
$x_t = x_{t-1} - \gamma g_t$
return x_t

Adagrad (3), like SGD, it is an optimizer that allows the use of a learning rate and a learning rate decay, which is responsible for decreasing the learning rate at each round.

Algorithm 3 Adagrad

Input: γ (lr), x_0 (params), $f(x)$ (objective), λ (weight decay), τ (initial accumulator value), η (lr decay), $state\_sum_0 = 0$
$g_t = \nabla_x f_t(x_{t-1})$
$\tilde{\gamma} = \gamma / (1 + (t-1)\eta)$
computing weight decay...
$state\_sum_t \leftarrow state\_sum_{t-1} + g_t^2$
$x_t = x_{t-1} - \tilde{\gamma} \frac{g_t}{\sqrt{state\_sum_t} + \epsilon}$
return x_t

Adam (4), takes into account the moving average of prime momentum (m) and second one (v) of the gradient.

4. Experiments

The experiments are conducted using a visual task based on image classification on CIFAR-10 datasets.

Clients Datasets. Each client is associated with an exclusive set of samples. All data are obtained by sampling the training dataset both in uniform and non-uniform process. The number of samples available for each client is assumed both as uniform and random depending on the type of

Algorithm 4 Adam

Input: γ (lr), β_1, β_2 (betas), x_0 (params), $f(x)$ (objective), λ (weight decay)

$g_t = \nabla_x f_t(x_{t-1})$

$\tilde{\gamma} = \gamma/(1 + (t-1)\eta)$

computing weight decay...

$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$

$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$

$\widehat{m_t} = m_t / \left(1 - \beta_1^t\right)$

$\widehat{v_t} = v_t / \left(1 - \beta_2^t\right)$

$x_t = x_{t-1} - \gamma \frac{\widehat{m_t}}{\sqrt{\widehat{v_t}} + \epsilon}$

return x_t

task. The random cardinality of each set is drawn from a *normal distribution* controlled with a *relative standard deviation* term (expressed as a fraction of the mean). The number of active clients in each round is assumed as a constant (valid in all the training phase) fraction of the whole group of clients, leading to a more sparse and realistic scenario.

Models Networks. The chosen architecture is a modified version of the original LeNet5 network [4] introduced in [2]. The in-order layers are:

- **First 2D Convolutional layer**: 3 inputs channels, 64 output channels, kernel size of 5.
- **First 2D Max Pooling layer**: square windows of size 2x2.
- **Second 2D Convolutional layer**: 64 inputs channels, 64 output channels, kernel size of 5.
- **Second 2D Max Pooling layer**: square windows of size 2x2.
- **3 Fully Connected layers**: 3 Linear transformations.

Dirichlet Prior. To synthesize a population of non-identical clients, for each client we draw a class distribution $q \sim Dir(p\alpha)$ from a Dirichlet distribution [1], where p characterizes a prior class distribution over N classes, and $\alpha > 0$ is a concentration parameter controlling the identicalness among clients: $\alpha \to 0$ produces sparse label while $\alpha \to \infty$ tends to produce more identical distributed classes. In our experiments α is selected with values 1, 10 and 100 (1). Higher value of α, are for simplicity redirected to the IID case.

Model Evaluation. Model's accuracy is measured by taking into account the server accuracy both on the

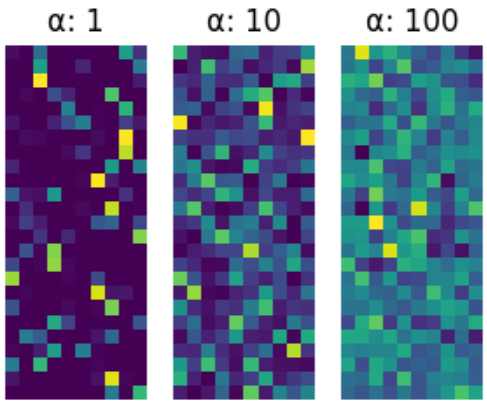


Figure 1. Classes distribution based on value of α. The example image shows 25 of 100 clients samples distribution per classes stacked on top of each other, all clients have a uniform number of samples. Sparsity can be seen by the dotted pattern assumed by the $\alpha = 1$ case.

training and test after a fixed amount of communication rounds. The considered metrics are the *maximum accuracy* reached as well as the *stability* and the *convergence time*.

4.1. FedAVG, FedIR and FedVC

Preliminary Federated Parameter Tuning. The experiment aims at finding a setting for the next FedAVG tasks. The number of clients is fixed to 50, with only 75% of them randomly selected at each round. Client and Server *batch size* is 64. Client has a standard SGD optimizer with a *learning rate* set to 0.01 and *momentum* to 0.9. The first tuned hyper-parameter is the local epochs number E for each client update. Results are visible in Figure 2. The general accuracy does not undergo variations, all models converge to the same accuracy. The convergence time of the model decreases as the number of epochs grows. The two best performances are mostly similar and, due to the excessive updating time for higher local epochs, $E = 2$ is used for all the next experiments.

IID and Non-IID comparison. In this setting we compare different values of α with the IID scenario. The results, in Figure 3, show a steady decline in performance as the α value decreases, also the convergence time increase as long as α tends to zero.

Federated Averaging vs Importance Reweighting. In this setting we compare different values of α. This setting is designed for proving the higher efficiency of FedIR in respect to FedAVG in case of non-identically distributed clients. The results, visible in Figure 4

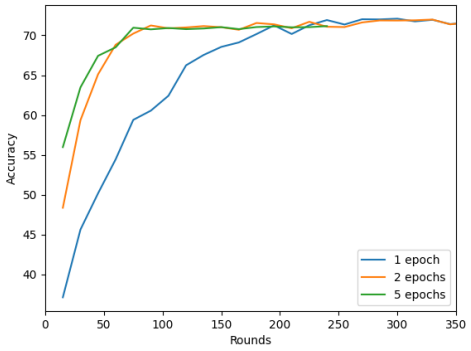


Figure 2. Models test accuracy based on number of rounds for different number of local epochs E.

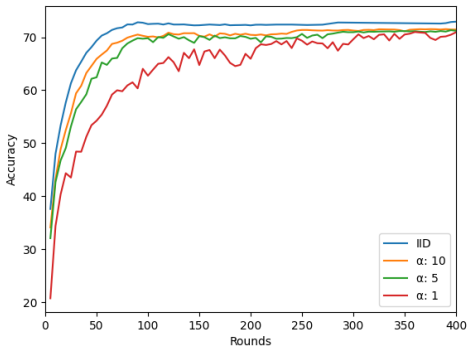


Figure 3. Model's test accuracy based on the number of rounds for different values of Dirichlet α and IID case.

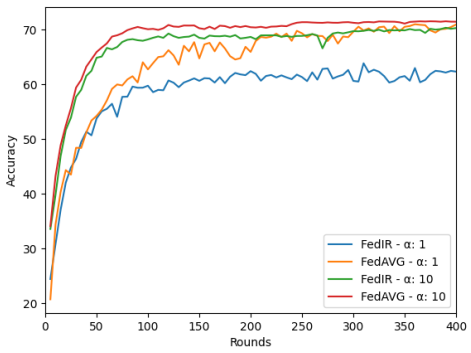


Figure 4. Model's test accuracy with respect to the number of rounds for different values of Dirichlet α.

deny the expected improvements for FedIR, which scores less than the standard method in every setting.

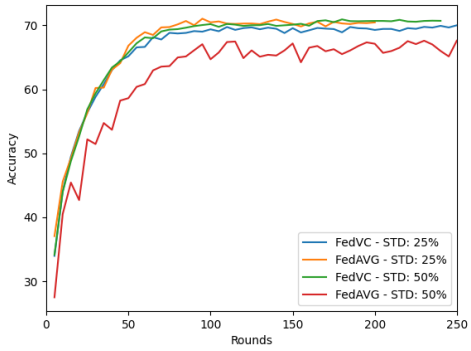


Figure 5. Models test accuracy based on number of rounds for different values of standard deviation STD.

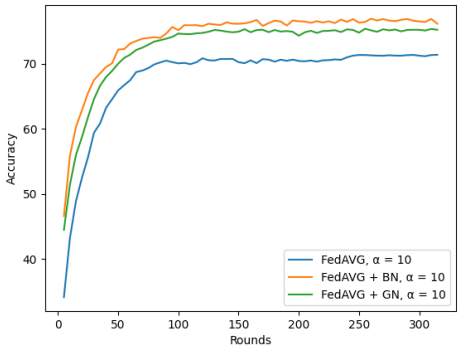
Federated Averaging vs Virtual Clients The setting compares different value of an STD parameter which represent the standard deviation, expressed in percentage of the average number of samples per each client. This setting is designed for proving the higher efficiency of FedVC in respect to FedAVG case of imbalanced clients' samples. A client with an overly large training dataset will take a long time to compute updates and takes significantly more local optimization steps than another with fewer training examples, creating a bottleneck in the federated learning round. The results, visible in Figure 5 show a consistent improvement even with high level of disparity (STD: 50%) for the FedVC accuracy, while the convergence time is unchanged.

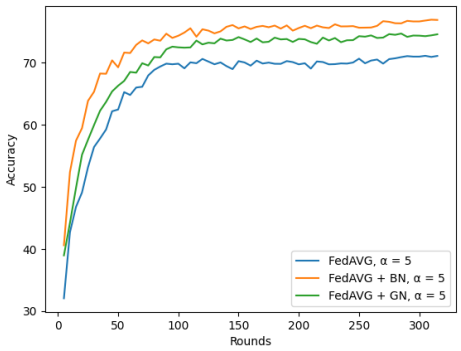
4.2. Normalization Layers

We try both Batch Normalization and Group Normalization after each convolutional layer of our LeNet5 in the centralized and federated settings. The Group Normalization is done dividing the 64 channels of the convolutional layers in 8 groups.

Federated Setting. To evaluate the effectiveness of normalization layers we test different values of class distributions among clients ($\alpha = [10, 5, 1]$). We test a typical scenario with 2 local epochs and 75% of total clients at every round. As we can see in 6 Group Normalization is effective only with more uniform class distribution. Batch Normalization overall has the best accuracy (on average 5% more than standard LeNet5). However, for very different class distribution (α=1) normalized models tend to have a very unstable behaviour compared to the original model. This problem will be addressed introducing the Adapted Federated Optimizations.

Centralized Setting. The same test has been done in the centralized settings, using same hyper-parameters used





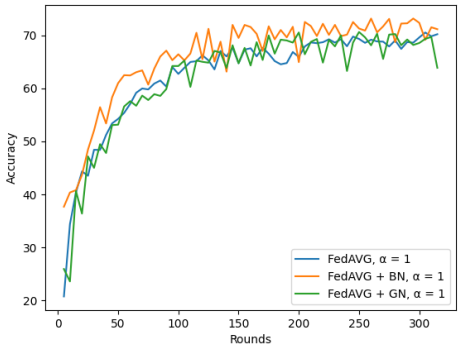


Figure 6. Test accuracy with $\alpha = [10, 5, 1]$. Comparison between original LeNet5 and modified versions with Batch and Group Normalization

in federated scenario (learning rate, momentum, batch size and no step-down policy in the optimizer) for a fair comparison. However, this setup is not optimal for this setting and higher accuracy can be reached with more fine-tuning.

Figure 7 shows that Batch Normalization is again the most effective with 80% accuracy on the test set, but the Group Normalization follow up close at 79%. The original LeNet5 is the worst with 77% accuracy.

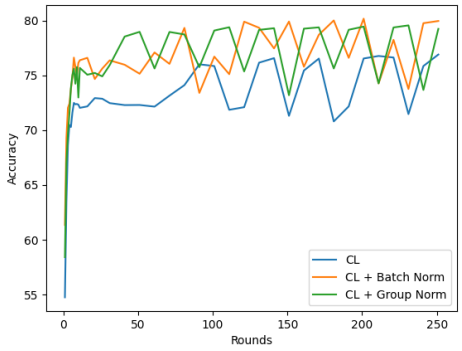


Figure 7. Test accuracy on centralized setting. Comparison between original LeNet5 and modified versions with Batch and Group Normalization

4.3. Adaptive Federated Optimization

Federated Averaging vs Adaptive Federated Optimization. We have tested different learning rates per each model. In general, lower learning rates bring more robust models but the convergence time increases. Below, we propose a comparison between the traditional federated algorithm and some adaptive federated optimization alternatives that performed well with our dataset and distribution.

FedOPT Hyper-parameters Selection All the experiments below were executed with the following parameters:

- FedAVG [LeNet5 & LeNet5 + BN]
  - $\mu(momentum) = 0$
  - $\eta(learning\ rate) = 1$
- FedOPT with SGD [LeNet5 & LeNet5 + BN]
  - $\mu(momentum) = 0.9$
  - $\eta(learning\ rate) = 0.1$
- FedOPT with Adagrad [LeNet5]
  - $\mu(momentum) = 0.9$
  - $\epsilon = 10^{-3}$
  - $\gamma(learning\ rate) = 0.001$

- FedOPT with Adagrad [LeNet5 + BN]
  - $\mu(momentum) = 0.9$
  - $\epsilon = 10^{-3}$
  - $\gamma(learning\ rate) = 0.01$
  - $\eta(learning\ rate\ decay) = 0.01$
- FedOPT with Adam [LeNet5 & LeNet5 + BN]
  - $\beta_1 = 0.9$
  - $\beta_2 = 0.99$
  - $\epsilon = 10^{-3}$
  - $\eta(learning\ rate) = 0.0005$

The accuracy on the training and test set are visible in Figure 8. Considering these results, we can confirm that, with an adaptive server optimizer, we can reach a more stable convergence especially in the last rounds.

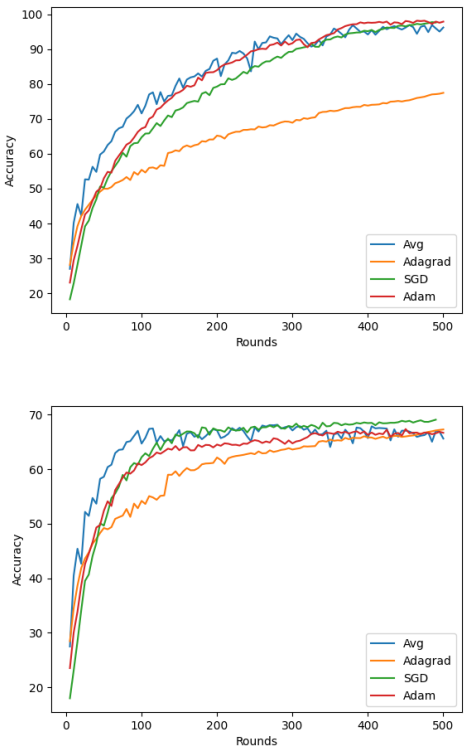


Figure 8. Model train and test accuracy with respect to the number of rounds per different server optimizers [LeNet5]

We've reproduced the same experiment also for the LeNet5 with the batch layers 9. The results are similar: at the end, the accuracy of the model is much less noisy than the ones using standard FedAVG.

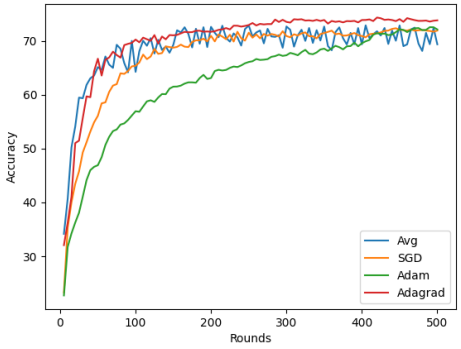


Figure 9. Model test accuracy with respect to the number of rounds per different server optimizers [LeNet5 + BN]

With these results we thus confirm that the Adaptive Federated Optimization can reach more robust models compared to the standard Federated Averaging and can be also easily extended with techniques to schedule the learning rate and with a variety of adaptive optimizers, in this way it should be possible to achieve a faster convergence, while ensuring a stronger model stability.

5. Conclusion

In this paper, we compared the standard optimization algorithms for the federated learning scenario, resulting in slight improvements in efficiency in different settings. We have provided further improvements by applying additional changes in the model structure using batch normalization. Finally, we have shown that adaptive optimizers can make improvements for FL convergence and limit the noisiness with clients having extremely unbalanced class distributions. Future works could bring to light further adaptive optimizers and extend their application to real world datasets.

References

[1] Tzu-Ming Harry Hsu, Hang Qi, and Matthew Brown. Measuring the effects of non-identical data distribution for federated visual classification. *arXiv preprint arXiv:1909.06335*, 2019. 4

[2] Tzu-Ming Harry Hsu, Hang Qi, and Matthew Brown. Federated visual classification with real-world data distribution. In *European Conference on Computer Vision*, pages 76–92. Springer, 2020. 1, 2, 4

[3] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015. 2

[4] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. 4

[5] Xiaoxiao Li, Meirui Jiang, Xiaofei Zhang, Michael Kamp, and Qi Dou. Fedbn: Federated learning on non-iid features via local batch normalization, 2021. 2

[6] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pages 1273–1282. PMLR, 2017. 1

[7] Sashank Reddi, Zachary Charles, Manzil Zaheer, Zachary Garrett, Keith Rush, Jakub Konečnỳ, Sanjiv Kumar, and H Brendan McMahan. Adaptive federated optimization. *arXiv preprint arXiv:2003.00295*, 2020. 1, 2, 3