Optimization methods for Federated Learning

Larigauderie Léo Benjamin Paul Killian, Ivan Korovkin, Lan Huong Tran
Department of Computer Science, EPFL, Switzerland

***Abstract*—In the context of Federated Learning (FL), optimization methods help to achieve better results in less time and resource usage. In this project, we aimed to perform experiments on several popular optimization methods for training neural networks in a Federated setting. We found that momentum SGD, Adagrad and Adam outperform SGD in both IID and non-IID data settings.**

I. Introduction

The idea of the Federated setting is to eliminate centralized data processing. Instead of this, the FL approach exploits the processing resources of end-users, by leaving the data on individual devices and sharing the global model to participants. This model is then trained on user devices with locally stored data. Further, the trained weights are sent back to the centralized server, where they are aggregated to improve the global model. This iterative algorithm continues, allowing the global model to learn from different clients without direct data sharing.

In this project, we explored the performance of different 1st order gradient-based optimization methods in the FL setting. We considered the Stochastic Gradient Descent (SGD) method [1] as our baseline. The performance of SGD was compared with the momentum variant of the SGD [2], Adam [3] and Adagrad [4] optimizers. This was done by comparing the performance of a Convolutional Neural Network (CNN) classifier we designed on the CIFAR-10 dataset [5] in the FL setting. Moreover, we tested the performance for two data repartition scenarios of independent and identically distributed (IID) and non-independent and identically distributed (non-IID) data on the client devices.

II. Optimizers

Considering an image classification problem, in which a model tries to predict the label $y \in R^m$ from an input image $x \in R^d$. The dataset D consists of n (image,label) pairs denoted as $z_i = (x_i, y_i)$, where i ranges from 1 to n. One commonly used approach is to employ gradient descent to minimize a loss function. The weights w of the model are iteratively updated using the following equation:

$$w_{t+1} = w_t - \gamma \frac{1}{n} \sum_{i=1}^{n} \nabla_w l(f_w(z_i))$$

Here, γ represents a reasonable learning rate. Variants of gradient descent exist and have been studied, with convergence properties demonstrated in practical applications.

A. SGD

In SGD, instead of computing the gradient on the full dataset D, each iteration estimates the gradient on a randomly chosen batch $z_t \in J$. The update rule becomes:

$$w_{t+1} = w_t - \gamma \frac{1}{|J|} \sum_{i=1}^{|J|} \nabla_w l(f_w(z_i))$$

where $|J| << n$. The convergence of stochastic gradient descent is proven under the conditions of decreasing learning rates that satisfy $\sum_t \lambda_t^2 < \infty$ and $\sum_t \lambda = \infty$.

B. Momentum

Momentum, introduced as a variant of SGD, addresses challenges such as saddle points and sharp curvatures while accelerating convergence [2]. It is based on the concept of Nesterov acceleration [6]. The update rule for momentum can be calculated as:

$$m_t = \beta_{t-1} + (1 - \beta) g_t$$

$$w_{t+1} = w_t - \lambda m_t$$

where $g_t = \nabla_w l(f_w(z_i))$ represents the gradient descent of an arbitrary training point z_t.

C. Adagrad

Adagrad [4] introduces adaptive scaling of learning rates for each parameter dimension, unlike stochastic gradient descent and momentum methods which use a fixed learning rate throughout training. Intuitively, it performs smaller updates for frequently occurring parameters and larger updates for infrequently occurring ones. Adagrad is particularly effective in handling sparse data and is more robust than SGD. However, it suffers from the issue of a vanishing learning rate due to the accumulation of squared terms in the denominator. The update is as follows:

$$[G_t]_i = \sum_{s=0}^{t} ([\nabla [g_s]_i])^2 \forall i$$

$$[w_{t+1}]_i = [w_t]_i - \frac{\lambda}{\sqrt{[G_t]_i}} [g_t]_i \forall i$$

where $g_t = \nabla_w l(f_w(z_i))$ is gradient descent of arbitrary training point z_t and $[x]_i$ denotes dimension i of vector x.

D. Adam

Adam is the improved variant of AdaGrad [3], aiming to overcome the weakness of learning rate vanishing, to handle very sparse and noisy gradients. Adam utilizes both concept of momentum term (momentum SGD) and adaptive learning rate (Adagrad). It has the advantages of intuitive interpretation and considerably little hyperparameter tuning. The update rule using Adam optimizer:

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$$
$$[v_t]_i = \beta_2 [v_{t-1}]_i] + (1-\beta_2)([g_s]_i)^2 \forall i$$
$$[w_{t+1}]_i = [w_t]_i - \frac{\lambda}{\sqrt{[v_t]_i}} [m_t]_i \forall i$$

where $g_t = \nabla_w l(f_w(z_i))$ is gradient descent of arbitrary training point z_t and $[x]_i$ denotes dimension i of vector x.

III. Experiments

To estimate the performance of the presented optimizers for federated learning, we propose to train a classifier on the CIFAR10[5] dataset, for both the IID and non-IID settings. We use Federated Averaging[7] to merge clients' parameters.

A. Network Architecture

We designed a classifier for CIFAR10 composed of a convolutional feature extractor with ReLU activations, followed by a fully connected layer. We follow each convolutional layer with a batch normalization layer to speed up and stabilize training by mitigating covariate shift[8]. We also use Dropout to mitigate overfitting by reducing co-adoption of units[9]. Specifically, we drop neurons with 0.5 probability right before the fully connected layer, which occurs after all batch normalization layers to circumvent variance shifts resulting from dropping neurons before batch normalization layers[10].

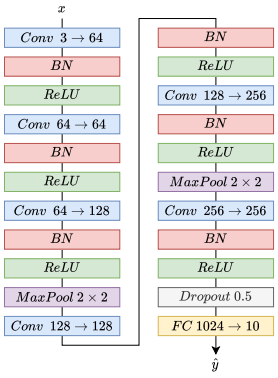

Figure 1. Our classifier, which takes as input a 32x23 RGB image x and outputs the estimated label vector $\hat{y}$. Each convolutional layer has a 3x3 kernel. We do not introduce any padding in convolutional layers

B. Data repartition among clients

We run all of our experiments with $C = 8$ clients. In federated learning, 2 cases are distinguished for data repartition among clients :

- IID, where all clients' data distribution is equal. In the case of labeled image data, for all client i, $(x, y) \sim P_i = P$ for some distribution P.
- non-IID, where this is not the case. There are multiple cases that lead to non-IID data[11], here we will focus on having the repartition of classes differ in each client.

To simulate the non-IID setting, we sample for each class what proportion of data samples from the original dataset will be assigned to each client according to the Dirichlet distribution as suggested in [12].

$$\mathbf{p}_i \sim Dir(\alpha)$$
$$\alpha \in R^C$$

where p_{ij} is the proportion of datasamples with label i assigned to client j. The choice of α affects the sparsity of the proportion vector. With $\alpha = 0.1$ we ensure most labels will have their corresponding data samples concentrated on a few samples, thus ensuring varied data distributions across all clients.

For the IID setting, we set $\alpha = 10^9$ which ensures that for each label, corresponding samples are evenly distributed between clients, thus resulting in each client having the same data distribution: that of the original dataset.

C. Hyperparameter tuning

In order to achieve better results with each optimizer, we performed hyperparameter tuning. To do so, we performed a 5-fold cross-validation (CV) for several learning rates (lr) for all optimizers, beta values (β_1, β_2) for Adam, and the momentum term for momentum SGD.

The results for several learning rates are represented in Table I in IID acc./non-IID acc. format for logging accuracy.

1) SGD: For SGD, we used 20 FL rounds for the IID setting, and 40 rounds for the non-IID setting as the non-IID setting experienced a lower convergence rate; both with 5 epochs per client per round. The experiment showed that $lr = 0.1$ is optimal for our network.

Optimizer \ LR	0.1	0.01	0.001	0.0001
SGD	**0.740** **0.573**	0.670 0.501	0.523 0.431	0.321 0.292
Momentum	**0.795** **0.610**	0.748 0.566	0.660 0.482	0.491 0.359
Adam	0.101 0.099	**0.788** **0.644**	0.746 0.585	0.640 0.293
Adagrad	0.753 0.530	**0.810** **0.608**	0.695 0.483	0.557 0.420

Table I
Learning rates for IID and non-IID data.

2) Momentum: For the cross-validation of Momentum SGD, we tuned the learning rate with fixed momentum=0.9 SGD using 15 FL rounds and 5 epochs per client per round. The results are summarized in Table I, which reveals that $lr = 0.1$ is the optimal learning rate in both scenarios. We further tuned the momentum $\in \{0.50, 0.90, 0.95\}$. The results, as presented in Table II, indicate that for the IID setting, momentum $= 0.90$ resulted in a higher validation score. Conversely, for the non-IID setting, momentum $= 0.50$ led to a better result.

Momentum	**0.5**	**0.95**	**0.99**
IID/Non-IID Acc.	0.766/**0.652**	**0.795**/0.610	0.777/0.560

Table II
MOMENTUM CROSS-VALIDATION FOR IID AND NON-IID DATA.

3) Adagrad: In our Adagrad experiment, we use the same setup as SGD in section III-C1 and received optimal result with $lr = 0.01$ for both IID and non-IID settings.

4) Adam: For Adam's optimizer CV, we trained our model on 10 FL rounds, with 5 epochs per client. The results were similar for $lr = 0.01$ and $lr = 0.001$, therefore, we have decided to implement the additional round of CV training for $lr = 0.005$, which showed the best result of 0.799/0.662. Moreover, we fine-tuned the (β_1, β_2) values for Adam. At the lr tuning, we used a default value of $(\beta_1, \beta_2) = (0.9, 0.999)$ for these attributes. However, we wanted to get insights into the accuracy improvements by choosing different sets of values, which were selected according to a different research paper [13]. The results for it are represented in Table III. For IID data, the model outperforms default values of $(\beta_1, \beta_2) = (0.9, 0.999)$ with the following parameters: $(\beta_1, \beta_2) = (0.5, 0.95)$, which we will use for final training. Moreover, it can be seen that the model shows its best performance for non-IID with $(\beta_1, \beta_2) = (0.95, 0.99)$, however, it's the same value as for the default case, so we decided to keep the original version.

β_2 \ β_1	**0.5**	**0.95**	**0.99**
0.5	0.802 / 0.472	0.098 / 0.099	0.099 / 0.100
0.95	**0.804** / 0.542	0.797 / 0.529	0.734 / 0.099
0.99	0.802 / 0.575	0.794 / **0.665**	0.765 / 0.322

Table III
ADAM'S BETAS FOR IID AND NON-IID DATA.

D. Results

For IID, in terms of training, both Adam and Momentum SGD demonstrated faster convergence after approximately 15 FL rounds, followed by SGD and Adagrad, which converged after around 25 and 40 epochs respectively. The training loss and accuracy graphs for all four optimizers exhibited stability and smoothness. Regarding test accuracy, Adam achieved the highest performance, reaching around 0.83, slightly surpassing Adagrad and Momentum SGD. In contrast, SGD resulted the worst test accuracy, yielding only around 0.76.

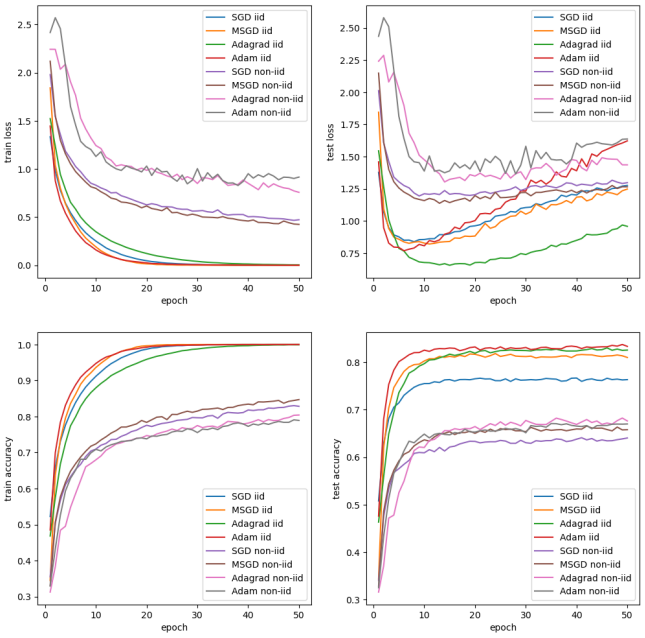

Figure 2. Train loss, Test loss & Train accuracy, test accuracy per epoch for SGD, momentum SGD (MSGD), Adagrad, Adam in the iid and non-iid settings

In comparison, for the non-IID setting, it was observed that all trainings exhibited a rougher and non-smooth evolution. After approximately 20 epochs, SGD and momentum SGD achieved significantly lower training loss compared to Adagrad and Adam. In terms of test accuracy, Adagrad achieved the highest accuracy score of 0.68, surpassing both Adam and momentum SGD with scores 0.67 and 0.66 respectively. All SGD variants performed significantly better than baseline SGD, which achieved roughly 0.64 accuracy. However, Adagrad had the lowest convergence, whereas Adam experienced the highest rate. The overfitting patterns were consistent with the convergence rate of each method. Although all optimizers started overfitting the dataset after 20 epochs, Adam's test loss increased drastically after 10 epochs.

IV. CONCLUSION

In this project, we explored the applicability of different optimizers for Federated Learning settings. For this, we have created a CNN which we were training on the CIFAR-10 dataset. The performance of the algorithm was tested in both IID and non-IID data between 8 clients. For the IID case, the best performance in terms of accuracy and convergence speed was shown by the Adam optimizer. In the non-IID scenario, the Adagrad optimizer slightly outperformed others, however, it was Adam optimizer that converged faster. In conclusion, we state that all tested optimizers (Adam, Momentum, Adagrad) outperformed the baseline SGD method in terms of accuracy in both scenarios.

REFERENCES

[1] H. Robbins and S. Monro, "A Stochastic Approximation Method," *The Annals of Mathematical Statistics*, vol. 22, no. 3, pp. 400 – 407, 1951. [Online]. Available: https://doi.org/10.1214/aoms/1177729586

[2] N. Qian, "On the momentum term in gradient descent learning algorithms," *Neural Networks*, vol. 12, no. 1, pp. 145–151, 1999. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0893608098001166

[3] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[4] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of Machine Learning Research*, vol. 12, no. 61, pp. 2121–2159, 2011. [Online]. Available: http://jmlr.org/papers/v12/duchi11a.html

[5] A. Krizhevsky, V. Nair, and G. Hinton, "Cifar-10 (canadian institute for advanced research)." [Online]. Available: http://www.cs.toronto.edu/~kriz/cifar.html

[6] Y. Nesterov, "Gradient methods for minimizing composite functions," *Mathematical programming*, vol. 140, no. 1, pp. 125–161, 2013.

[7] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," 2023.

[8] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *CoRR*, vol. abs/1502.03167, 2015. [Online]. Available: http://arxiv.org/abs/1502.03167

[9] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, no. 56, pp. 1929–1958, 2014. [Online]. Available: http://jmlr.org/papers/v15/srivastava14a.html

[10] X. Li, S. Chen, X. Hu, and J. Yang, "Understanding the disharmony between dropout and batch normalization by variance shift," *CoRR*, vol. abs/1801.05134, 2018. [Online]. Available: http://arxiv.org/abs/1801.05134

[11] P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, K. A. Bonawitz, Z. Charles, G. Cormode, R. Cummings, R. G. D'Oliveira, S. E. Rouayheb, D. Evans, J. Gardner, Z. Garrett, A. Gascón, B. Ghazi, P. B. Gibbons, M. Gruteser, Z. Harchaoui, C. He, L. He, Z. Huo, B. Hutchinson, J. Hsu, M. Jaggi, T. Javidi, G. Joshi, M. Khodak, J. Konečný, A. Korolova, F. Koushanfar, S. Koyejo, T. Lepoint, Y. Liu, P. Mittal, M. Mohri, R. Nock, A. Özgür, R. Pagh, M. Raykova, H. Qi, D. Ramage, R. Raskar, D. Song, W. Song, S. U. Stich, Z. Sun, A. T. Suresh, F. Tramèr, P. Vepakomma, J. Wang, L. Xiong, Z. Xu, Q. Yang, F. X. Yu, H. Yu, and S. Zhao, "Advances and open problems in federated learning," 2019. [Online]. Available: https://arxiv.org/abs/1912.04977

[12] M. Luo, F. Chen, D. Hu, Y. Zhang, J. Liang, and J. Feng, "No fear of heterogeneity: Classifier calibration for federated learning with non-iid data," 2021.

[13] V. Felbab, P. Kiss, and T. Horváth, "Optimization in federated learning," in *Conference on Theory and Practice of Information Technologies*, 2019.