Federated Learning

Jacob Pollard
Department of Electrical and Computer Engineering
University of California, San Diego
San Diego, CA 92093
jpollard@ucsd.edu

ECE 273, Spring 2021

Abstract

Federated Learning (FL) shows great promise in resolving a number of problems in ML. However, it also introduces a vast array of unique challenges not seen in other frameworks. Here we will give an overview of FL, and explore some of the new technical challenges that are presented by it. The objective here is to explore recent work which serves two purposes. First, to expose some of the issues with FL, as well as to present solutions to some of its problems

1 Background

Federated Learning is a rising framework that seeks to train machine learning models using a new paradigm. The fundamental difference between federated learning and other methods is that the central model has no access to the data on which it is being trained. While a federated model introduces a wide number of technical challenges, it also provides a number of benefits. For example, over networks with limited capacity, it may not even be desirable to transfer the full data over to a central server.[1] Since Federated Learning also keeps data opaque to the central server, it is also desirable in situations where privacy is a priority, such as with medical data, or even personal mobile phone data.

1.1 Typical Federated Learning Workflow

Lets review Federated Stochastic Gradient Descent (SGD). The model is similar to typical SGD, but distributed across workers [2]. Let there be k workers with $\mathcal{D}_1, ..., \mathcal{D}_k$, the dataset stored on each worker and $p_1, ..., p_k$ hyperparameter weights assigned to each worker. Typical values for these p_i are $\frac{1}{k}$, or $\frac{1}{|\mathcal{D}_i|}$. These p_i weight each workers impact on the master update. The goal is to learn some weight vector w in order to minimize some loss function $\mathcal{L}(w; \mathcal{D})$. After initializing our hyperparameters, and the weight $w = w_0$, the following is repeated until convergence. First, the master broadcasts the current weight w_t to all workers. Now, each worker will compute it's own update to the gradient $w_{t+1}^i := w_t - \alpha \nabla \mathcal{L}(w_t; \mathcal{D}_i)$ Finally, all weights $w_{t+1}^1, ..., w_{t+1}^k$ are sent back to the master, which computes the new weight $w_{k+1} := \sum_{i=1}^k p_i w_{t+1}^i$, and the process begins again. The algorithm is given below.

Algorithm 1: Federated SGD

```
Initialize weight: $w = w_0$;
Worker weights: $p_1, ..., p_k$;
Worker data: $\mathcal{D}_1, ..., \mathcal{D}_k$;
Learning rate $\alpha$;
while not converged do
    Master broadcast $w_t$ to all workers;
    foreach worker $i = 1, ..., k$ do
        $w_{t+1}^i := w_t - \alpha \nabla \mathcal{L}(w_t; \mathcal{D}_i)$;
        Worker broadcast $w_t^i$ master;
    end
    Master calculate new weight $w_{k+1} := \sum_{i=1}^k p_i w_{t+1}^i$;
end
```

1.2 Applications

One classically referenced example of federated learning is in the case of training an auto complete model for smartphones. It is desirable to leave people's typing data on their device in order to preserve their privacy, and so this is a natural case for Federated Learning. As with most all Federated Learning use cases, this one too has its unique challenges, two of which we will discuss. First, the data may not be identically distributed. People tend to type differently, and language use can vary widely across one country (for example, compare the south eastern and south western US). Second, people have vastly different hardware on their phones, and therefore there are unique challenges in computing model updates. This is a perfect example of where FL is heterogeneous in both computation and data [3].

Another common domain for FL is medicine [4]. Medicine is of great interest with FL since medical data is so private, and highly regulated. In this case, training need not be done on mobile devices, but privacy is of higher importance than in other cases. Indeed the prospect of using ML to improve medicine is an exciting one.

2 Experimental setup and overview

For the experiments, we use a simple model and dataset. The reason for this is two fold. Firstly, since many trials will be run, we desire a model that has a reasonably fast convergence time in general. Secondly, creating and tuning a complex model would detract from our ability to explore the peculiarities of that model in the federated setting.

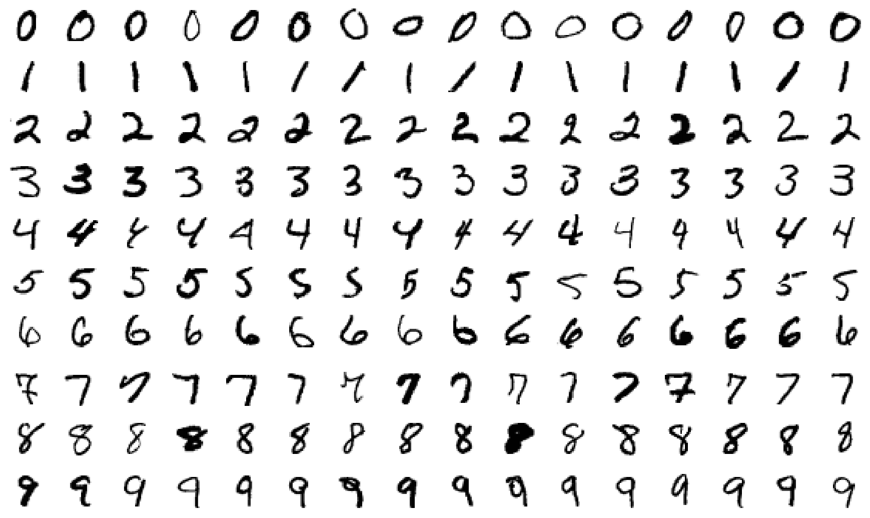
2.1 Model

In all sections our model is a simple convolutional neural network with 2 convolutional layers. In all cases, will use SGD as our optimization algorithm, and categorical cross entropy as our loss function. In sections 2 and 3, we use ReLU as the activation function, while in section 4, we switch this to a sigmoid. This is because the experiments in section 4 requires the existence of second derivatives, as will be discussed. For sections 2 and 3, the models are implemented in tensorflow[5] and in section 4 we use pytorch [6] in order to utilize the code provided by the authors of the original work.

2.2 Dataset

Our dataset is the classic MNIST[7] dataset. This is a dataset of small (28x28) handwritten digits. We wish to classify these digits into classes which are the numbers that they represent.

Figure 1: Samples from the dataset



3 Evaluating a simple model using Federated Averaging

3.1 Overview

In this section we will compare the performance of the model in both the federated and centralized settings. Indeed, we will use the exact algorithm as discussed in section 1, as referenced in algorithm 1.

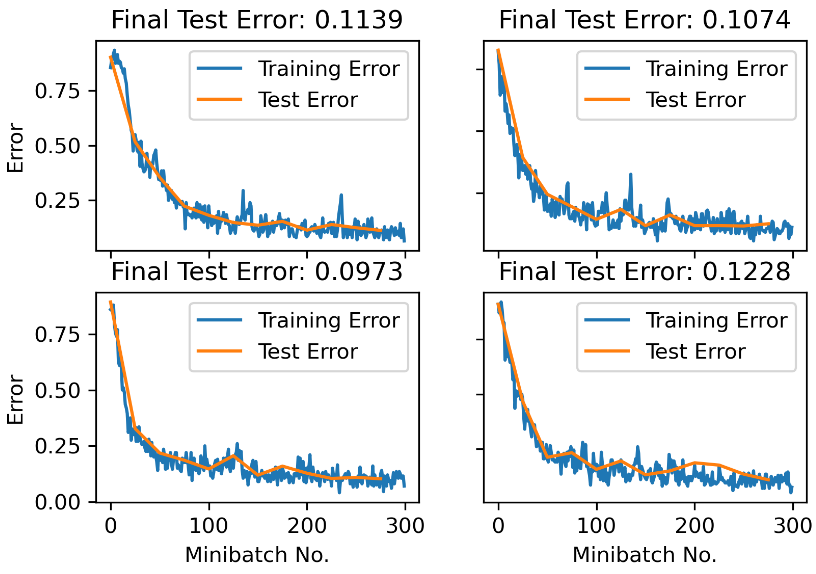
3.2 Methodology

We first will run SGD in the centralized setting, then we will compare the results with those in the Federated setting. Using the Federated Averaging schema, we run experiments with $k = 5, 10, 20, 25$ workers. We'll then compare the results with those of the centralized model. In this section, we have a minibatch size of $b = 200$.

3.3 Results & Discussion

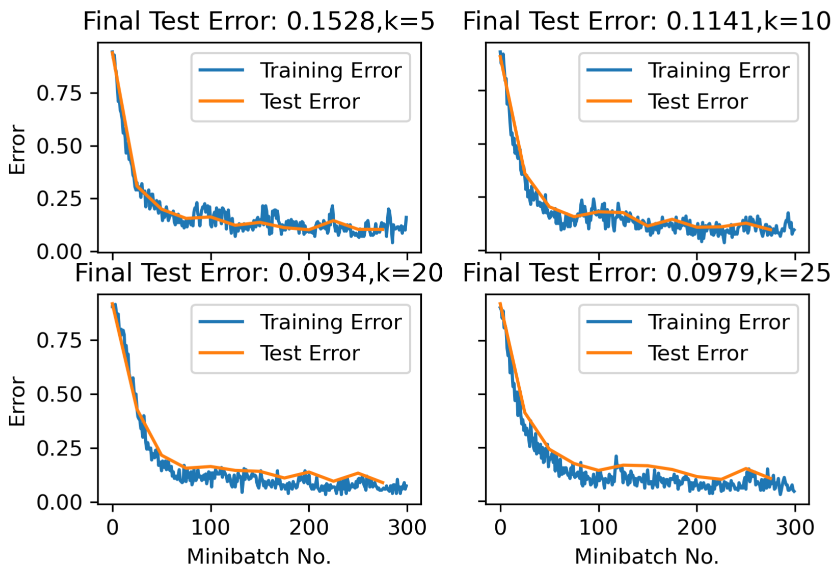
For Centralized SGD, the algorithm converges in a single epoch, and a test error rate of about 10% is achieved.

Figure 2: Centralized SGD



In the case of the Federated system, we can see that the test error at first appears to be quite similar. In truth, the situation is a bit more complicated.

Figure 3: Federated SGD



Since the batch size $b = 200$ is the same for all k workers, much more computation in being done to achieve the same thing as in the centralized system, to achieve approximately the same result. This is because, at each step, the federated workers calculate their gradients for their minibatches, each of size b, then the master accumulates and takes the average of them for the global model update. Therefore, at each round kb examples are viewed. Therefore, although each Federated trial shows the algorithm converging at approximately the same rate, in reality k times more computation is required to achieve the same goal. Further, this does not include the cost of transmitting information, which in the federated system especially, is of high importance as updates must often occur on constrained networks.

4 Evaluating and improving the robustness of Federated Learning

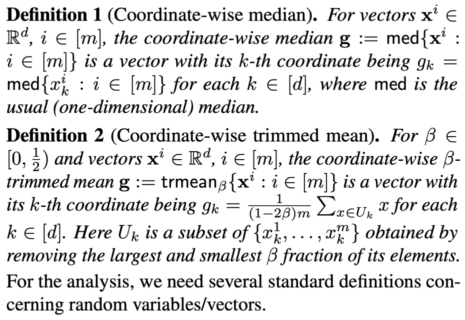
Now we will evaluate strategies to mitigate the effect of malicious workers. Federated systems are highly decentralized, and in many cases, there is little or no control over the behavior of workers. Consider the case of training a text correction app as discussed earlier. In this situation, it is possible that a user who wishes to corrupt the model might input a large volume of text into their phone which is meant to confuse the model. Or, consider a user who frequently uses foul language. In each of these cases, the designer may wish to eliminate the possible impact of these user's updates on the model.

4.1 Methodology

In this section we follow the work in [8]. We consider a situation where a proportion, α of the users are malicious, and send tainted updates to the master. This will be implemented as follows. Each training example has an integer label $l = 0, ..., 9$. The malicious users will instead train with labels $l' = 9 - l$, and send their corresponding updates.

The goal of this section is to study how to maintain a robust model in the presence of these malicious updates. The only adjustment that we make to Algorithm 1, is the method used in the aggregation of the weights to form the master update. Instead of taking the average, we will take the coordinate-wise $trimmedMean$ and coordinate-wise $median$, whose definitions are provided in figure 4 for completion. When choosing the number of malicious users, we take the $m = floor(k\alpha)$, for example if $k = 25, \alpha = .1$, we take $m = 2$.

Figure 4: Definitions, taken directly from [8]



Definition 1 (Coordinate-wise median). *For vectors* $\mathbf{x}^i \in \mathbb{R}^d$, $i \in [m]$, *the coordinate-wise median* $\mathbf{g} := \mathsf{med}\{\mathbf{x}^i : i \in [m]\}$ *is a vector with its k-th coordinate being* $g_k = \mathsf{med}\{x_k^i : i \in [m]\}$ *for each* $k \in [d]$, *where* med *is the usual (one-dimensional) median.*

Definition 2 (Coordinate-wise trimmed mean). *For* $\beta \in [0, \frac{1}{2})$ *and vectors* $\mathbf{x}^i \in \mathbb{R}^d$, $i \in [m]$, *the coordinate-wise* β-*trimmed mean* $\mathbf{g} := \mathsf{trmean}_\beta\{\mathbf{x}^i : i \in [m]\}$ *is a vector with its k-th coordinate being* $g_k = \frac{1}{(1-2\beta)m} \sum_{x \in U_k} x$ *for each* $k \in [d]$. *Here* U_k *is a subset of* $\{x_k^1, \ldots, x_k^m\}$ *obtained by removing the largest and smallest* β *fraction of its elements.*

For the analysis, we need several standard definitions concerning random variables/vectors.

4.2 Results & Discussion

The Tables 1,2, and 3 give the test error rates with malicious users using the regular mean update, coordinate-wise trimmed mean, and coordinate-wise median respectively. We have bolded the **best** score for each respective category. We can see that the mitigation techniques do work (with the exception of a few outliers), and in the cases where the normal mean update achieves the best results, the difference is extremely small (considerably less than .01). We can see that the median tends to be the best performer overall. The gains made by these mitigation techniques are relatively mild, but the methodology is also extremely simple, which gives virtue to the method.

Num Workers: α	10	20	25
.05	X	.1272	**.1081**
.1	.1327	.1467	.1212
.2	.1845	**.1584**	.1906

Table 1: Test Error results with malicious users, using the regular mean update rule

Num Workers: α	10	20	25
.05	X	.1447	.1486
.1	**.0996**	**.1162**	.1754
.2	.1563	.1609	.1353

Table 2: Test Error results with malicious users, using coordinate-wise trimmed mean, $\alpha = \beta$

Num Workers: α	10	20	25
.05	X	**.0957**	.1091
.1	.1215	.2027	**.0998**
.2	**.1344**	.17	**.1209**

Table 3: Test Error results with malicious users, using coordinate-wise median

5 Evaluating the possibility of unintentional information capture

In this section we will explore training data acquisition by snooping on worker transmitted gradient updates. As discussed earlier, one reason that FL is used, is in the interest of protecting the privacy of the data. Initially, it would seem that transmitting gradient updates back to the master would not be risky, as the dataset is safely stored on the worker node, and as such cannot be intercepted. We will see that this is not the case.

5.1 Methodology

Here, we will follow the work in [9]. The algorithm which we will discuss can take the gradient update, as well as the worker weight state, and produce training data. First, the model gradient update, as well as the current weights must be intercepted. Then the algorithm proceeds as follows: first, the "data" (we'll refer to it as learned data) are initialized to Gaussian noise. After this, the derivative is taken on the model with respect to the weights and evaluated at the learned data (call this the learned gradient). Then, x and y are found such that the distance between the true gradient and learned gradient have a minimum distance. Then, the learned data are updated with the gradient of this distance, evaluated at the data point. This algorithm is given in Algorithm 2. The only caveat is that in practice, the batchsize here must be 1 or convergence requires a long amount of time.

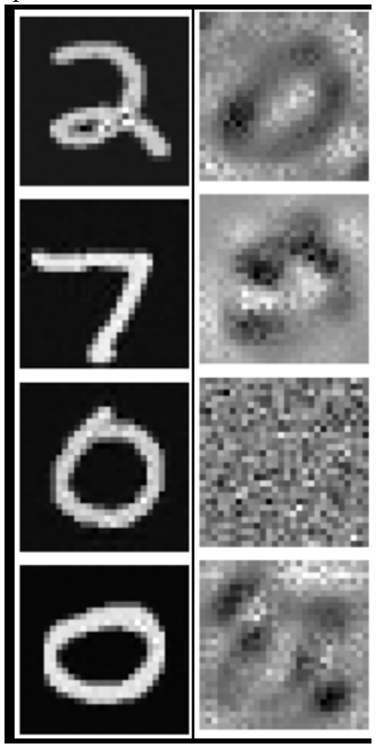
Algorithm 2: Deep Leakage from Gradients [9]

```
Input: $F(x; W)$ differentiable model;
$W_t$ model parameters;
$\nabla W_t$ gradient on training data;
Output: training data x,y;
Initialize by sampling: $x, y\ \mathcal{N}(0, I)$;
**for** $i = 1, ..., n$ **do**
    $\nabla W_i' := \partial l(F(x_i; W_t), y_i)/\partial W_t$;
    $G_i := ||\nabla W_i - \nabla W_t||_2^2$;
    $x_{i+1} := x_i - \alpha \nabla_{x_i} Gi$;
    $y_{i+1} := y_i - \alpha \nabla_{y_i} Gi$;
**end**
return $x_{n+1}, y_{n+1}$
```

5.2 Results & Discussion

We were able to reproduce a test data about 50% of the time. Qualitatively, it seems that successful reconstruction is more possible when the gradient is intercepted earlier in the training process. This may be because the gradient contains more information earlier on in training, as the model has more "room for improvement".

Figure 5: Samples of successful and unsuccessful recoveries



6 Conclusion

In this paper, we reviewed some basic applications and algorithms in Federated Learning. We studied some mitigation techniques which can be used to handle the presence of bad worker gradient updates. Additionally, we saw the feasibility of acquiring training data by intercepting gradient updates. These insights help to display the unique challenges, risks, and opportunities this emerging schema presents.

References

[1] Mohammed Aledhari, Rehma Razzak, Reza M. Parizi, and Fahad Saeed. Federated learning: A survey on enabling technologies, protocols, and applications. *IEEE Access*, 8:140699–140725, 2020.

[2] Ruixuan Liu, Yang Cao, Masatoshi Yoshikawa, and Hong Chen. Fedsel: Federated SGD under local differential privacy with top-k dimension selection. *CoRR*, abs/2003.10637, 2020.

[3] Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. Federated learning: Challenges, methods, and future directions. *IEEE Signal Processing Magazine*, 37(3):50–60, 2020.

[4] Theodora S. Brisimi, Ruidi Chen, Theofanie Mela, Alex Olshevsky, Ioannis Ch. Paschalidis, and Wei Shi. Federated learning of predictive models from federated electronic health records. *International Journal of Medical Informatics*, 112:59–67, April 2018.

[5] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[6] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[7] Yann LeCun and Corinna Cortes.

[8] Dong Yin, Yudong Chen, Ramchandran Kannan, and Peter Bartlett. Byzantine-robust distributed learning: Towards optimal statistical rates. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 5650–5659. PMLR, 10–15 Jul 2018.

[9] Ligeng Zhu, Zhijian Liu, and Song Han. Deep leakage from gradients. *CoRR*, abs/1906.08935, 2019.