On implementing custom training and aggregation strategies for federated learning with Flower

Sasi Kumar Murakonda

1 Introduction

The primary objective of this study is to analyse the flexibility and ease of customising the server strategy in Federated Learning when using the Flower Framework. To do so, we implemented five simple custom strategies for training and aggregation of models in a cross-silo federated learning setup. We present our experience implementing these strategies along with an empirical evaluation of how they performed in different settings for an image classification task.

The report is organised as follows: we briefly describe the strategies and explain our reasoning for picking these strategies in section 2. Then we present the implementation details of these strategies in section 3, followed by a presentation of the evaluation results in section 4. Finally, we conclude with a discussion of ideas that we couldn't dig into due to the limited time in section 5.

2 Strategies

2.1 LRDecay (Original Problem Statement)

This strategy sends different instructions to different sets of clients during the training process. More specifically, half of the clients train with a constant learning rate and the other half implement an exponential learning rate decay. It uses the standard FedAvg algorithm for aggregation of parameters at the server. The learning rate at round i for the clients implementing weight decay is given by:

$$lr_i = lr_0 \times D^{i-1} \tag{1}$$

where D is the decay factor - set to 0.99 in all our experiments.

2.2 Federated Learning meets Control Systems

~~*No plan survives contact with the enemy.*~~ *No theory survives contact with practice.*

I wanted to play around with the aggregate fit function in the strategy class and was searching for algorithms that were tested on actual data partitions from different organisations in the real world. FedCostWAvg [2] and FedPIDAvg [3] are the *only* works that I found mentioning any evaluation on real data and they **won the FETs challenge for brain tumor segmentation [6] in the category of effective weight aggregation methods in federated learning**.

Unfortunately, these works haven't received much attention and aren't even published in any conference. I speculate the reason could be a lack of rigor and potential privacy risks (more details on this in appendix A). But irrespective of these shortcomings, I personally feel that their attempt to connect federated learning with control theory [4] offers a novel perspective about the role of the server in FL and deserves more attention from the community.

2.3 FedCostWAvg [2]

When computing the aggregated parameters, FedAvg weighs the parameters from each client based on the number of examples at the client. FedCostWAvg is a slight modification of FedAvg where along with the number of examples, relative decrease in the loss function value at each client is also added in the weight.

$$M_{i+1} = \sum_{j=1}^{n} (\alpha \frac{s_j}{S} + (1-\alpha)\frac{k_j}{K}) M_i^j \tag{2}$$

Here n is the number of clients sampled in round i, s_j the size of the dataset at client j, and $S = \sum_j s_j$. The loss ratio factor is computed as:

$$k_j = \frac{c(M_{i-1}^j)}{c(M_i^j)}, K = \sum_j k_j \tag{3}$$

where $c(M_i^j)$ is the cost of local model with client j at step i.

The relative influence of number of examples compared to the decrease in loss values at the clients is controlled by the parameter α, set to 0.5 in all our experiments.

This was the winning solution at the FETs 2021 challenge [6] - an international challenge to develop algorithms for building a brain tumor segmentation model in the federated setting.

2.4 FedPIDAvg [3]

Inspired by the concept of a PID controller, this is a modified version of FedCostWAvg and was the winning solution at the FETs 2022 challenge. It uses a combination of loss differentials and integrals at the clients, along with the number of examples, to determine the weights for each client during parameter aggregation at the server.

$$M_{i+1} = \sum_{j=1}^{n} (\alpha \frac{s_j}{S} + \beta \frac{k_j}{K} + \gamma \frac{m_j}{I}) M_i^j \tag{4}$$

Here n is the number of clients sampled in round i, s_j the size of the dataset at client j, and $S = \sum_j s_j$. The loss differential factor is computed as:

$$k_j = c(M_{i-1}^j) - c(M_i^j), K = \sum_j k_j \tag{5}$$

where $c(M_i^j)$ is the cost of local model with client j at step i. Notice that instead of ratio as in FedCostWAvg, here difference of the cost functions is used.

The loss integral factor is computed as:

$$m_j = \sum_{l=0}^{5} c(M_{i-l}^j), I = \sum_j m_j \tag{6}$$

The relative weights of number of examples, loss differential, and loss integral terms are controlled by α, β, and γ, which sum to 1. They were set to 0.45, 0.45, and 0.1 in all our experiments.

$$\alpha + \beta + \gamma = 1 \tag{7}$$

Note: The equation computing m_j is wrongly stated in the original paper as $m_j = \sum_{l=0}^{5} c(M_{i-l})$, which means the loss sums are calculated over the aggregated model, rather than the local models. I don't think that was the intent of the original authors and the integral term should also be evaluated on the local models. This confusion led to a mix-up in my implementation as well, consuming a significant amount of my time and compute resources.

2.5 FedCostWAvgLRDecay

As the name suggests, this is just a combination of FedCostWAvg with LRDecay i.e., train half of the clients with a learning rate decay and use FedCostWAvg during parameter aggregation.

2.6 FedPIDAvgLRDecay

Similar to FedCostWAvgLRDecay, this is just a combination of FedPIDAvg with LRDecay i.e., training half the clients with a decaying learning rate and using FedPIDAvg at the server.

3 Experience implementing custom strategies with Flower

3.1 LRDecay

Thanks to the really intuitive setup of Flower and the multitude of already implemented strategies, this was pretty straightforward. I really liked the idea of having the on_fit_config_fn. I think it's a pretty neat way to send instructions that are same for all the clients. So, just to play around with it, I created a function that sends the number of epochs to train during each round. These configuration files were first imported from the parent class (FedAvg) and were then edited to append the learning rate instruction in the LRDecay class.

For determining the client split, I relied on the client ID being an odd or even number. As Flower automatically assigns integers in the range of number of clients as the client IDs, this works fine if no client ID names were specified when starting the simulation. If the client Ids were pre-stated as input to the simulation, then we might have to use a hash function or an explicit mapping to determine which half of the clients to send the instructions for implementing learning rate decay. Note that if we use a hash function, the clients will not be split into two halves of equal sizes and it isn't really that effective when the total number of clients is very small.

3.2 FedCostWAvg and FedPIDAvg

I had to make very simple changes to the aggregate_fit function in the FedAvg class to implement these aggregation functions. But given they require additional inputs from each client (beyond the number of examples), I implemented custom clients that send the required information in the metrics part of the fit results. I had to use a client directory to store the history of losses as Flower tears down the clients and doesn't maintain any state information in the simulation mode.

3.3 FedCostWAvgLRDecay and FedPIDAvgLRDecay

These just inherit the aggregate fit function from their corresponding aggregation algorithm strategy classes and configure fit function from the LRDecay strategy class.

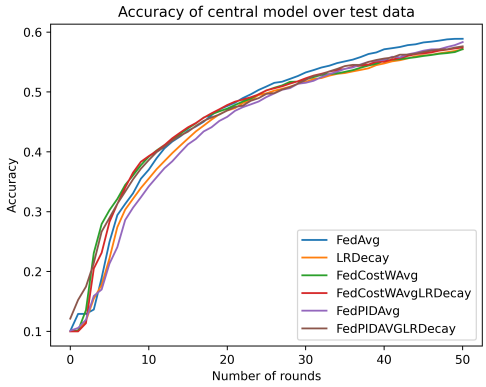
4 Performance evaluation of the strategies

Task: Image classification of the CIFAR-10 dataset with a very simple convolutional neural network - taken directly from the pytorch introductory tutorial.

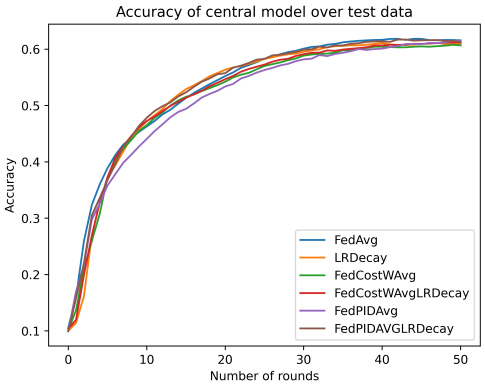
Data Partitioning: The data is partitioned using a custom partition function, where clients with an even number id get twice the examples as in the clients with an odd number id.

Performance criterion: Accuracy of the aggregated model on the server data (test data of CIFAR-10) after each round.

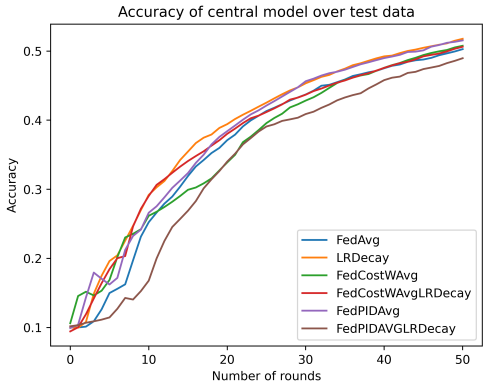
Some empirical observations:



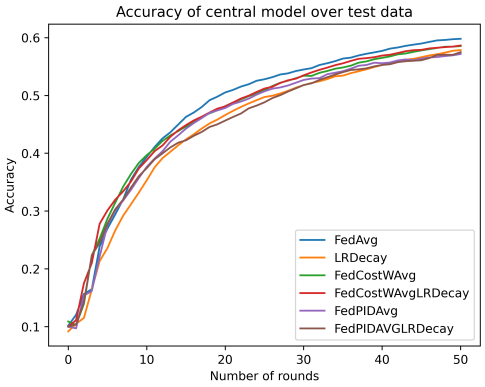
(a) 5 clients, 1 epoch, lr= 0.01



(b) 5 clients, 2 epochs, lr= 0.01

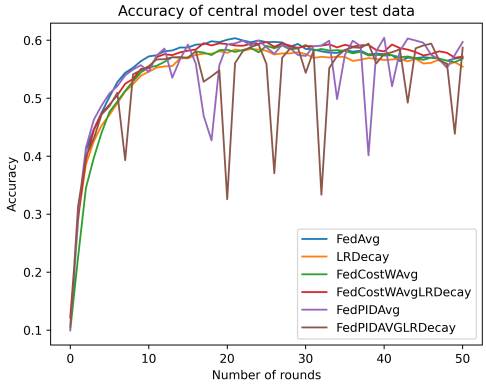


(c) 10 clients, 1 epoch, lr=0.01

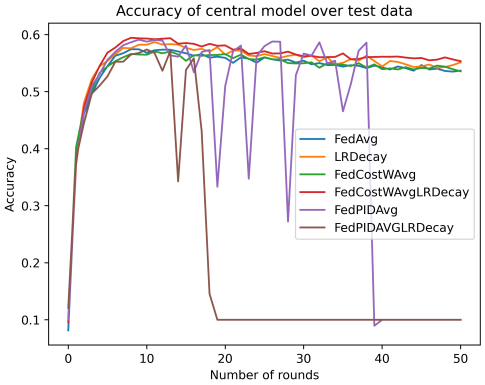


(d) 10 clients, 2 epochs, lr=0.01

Figure 1: These plots compare the performance of our custom server strategies in four different settings. Figures 1a and 1b show the results when there are 5 clients training at a learning rate of 0.01 for one epoch and two epochs respectively. Similarly figures 1c and 1d represent the setting with 10 different clients. All the models were trained for 50 rounds and with a weight decay factor of 0.99 when applicable. Notice in 1a and 1c that FedAvg catches up with the rest of strategies as the training proceeds whereas in 1b and 1d FedAvg seems to lead from early rounds itself. When the clients haven't fully learned the model i.e., when training with less number of epochs, cost function weighted strategies can help create better central models and accelerate the learning process. Whereas once the client models have learned to fit their data well and are in agreement, FedAvg (as expected) is the optimal aggregation strategy.



(a) 5 clients, 1 epoch, lr = 0.1



(b) 5 clients, 2 epochs, lr = 0.1

Figure 2: Here we compare the effect of using a large learning rate on all the strategies. As expected, during the initial phases of fast learning and agreement, FedAvg leads with all the other strategies being very close. The overfitting phase, where test accuracy stays flat or even goes down, reveals some unexpected patterns. First, it clearly highlights the stability issues with FedPIDAvg aggregation strategy. We provide more details about this phenomenon in Appendix A.2. The other surprising factor is the close to null effect of using learning rate decay compared to standard FedAvg, although FedCostWAvg with learning rate decay seems to fare better than plain FedCostWAvg.

1. Maximum test accuracy that could be achieved seems to be around 0.6 irrespective of what hyper-parameters and number of clients we choose - this is probably a limitation of the network size/structure that we have chosen.
2. The performance variations, at least for FedAvg and FedCostWAvg (with and without learning rate decay) aren't really that significant - it seems like the variance in performance across strategies isn't much larger than the variance in performance of any one strategy across different model initializations.
3. FedAvg performs as good as any algorithm, if the models from clients agree with each other (probably due to a large value for learning rate or training for a large number of local epochs).
4. FedCostWAvg and FedPIDAvg could help fasten the learning process compared to FedAvg during the initial phases of model disagreement across clients (due to partial learning because of training for less number of epochs or a smaller value of learning rate).
5. FedPIDAvg is a very unstable algorithm, especially once the model starts overfitting i.e., when the test accuracy starts to go down. Some comments about this in appendix A.2.

5 Discussion

Here are some ideas that I wanted to test but couldn't manage to because of the time limit.

5.1 Hyperparameter tuning in federated learning

Selection and tuning of hyperparameters (e.g., learning rate, batch size, number of epochs, etc) is one of the most challenging and expensive aspect of training machine learning models. In practice, setting these values is just based on heuristics and (well tested) folk knowledge. The problem statement to send different learning rates to different sets of clients got me thinking about the current state of the art in hyperparameter tuning for federated learning. After a brief search, I found this really interesting paper [1] that proposes FedEx, an approach to hyperparameter tuning at the clients that works well with FedAvg.

From an implementation perspective, the FedEx algorithm is really simple. All the clients maintain a prior probability distribution over the hyperparameters they want to test. After every

round, along with the updated model parameters, they receive an updated probability distribution over the hyperparameters. They sample the hyperparameters for that round from the updated probability distribution. I didn't understand the theoretical connections of this algorithm with neural architecture search and why their probability distribution update works. But as per the results in the paper, this approach performs really well for selecting the hyper parameters.

One caveat with the FedEx algorithm is ironically its own hyperparameters. Also, creating an appropriate set of hyperparameters to explore could be challenging, especially when it has a lot of continuous values (such as learning rate) instead of discrete values (number of epochs, batch size). Even with its limitations, I think this could be an interesting algorithm to implement and test.

5.2 Advanced partitioners for simulating real world data splits

Most works rely on custom partitions of data to evaluate the performance of algorithms for federated leaning. In order to simulate a non-iid setting, the typical approach seems to be to vary the number of examples at different clients and biased sampling of the data based on the labels to be predicted. Although these can offer non-iid datasets to an extent, they are far away being any representative of the data distributions across organisations in the real world. Advanced data partitioners to help model these differences could be very helpful to researchers in benchmarking and designing new training/aggregation strategies for federated learning.

One trick that I have been thinking about tinkering with is to train generative models of increasing complexity on the data and then use different distributions over these models for sampling the data at different clients. The process roughly looks like:

1. Given the dataset, train generative models of increasing complexity
2. For each client, select a distribution over these models for sampling the data.
3. When generating data at the client, first select the generative model and then sample.

The generative models could be Bayesian networks with constraints on the structure (e.g., first model allows only marginals, the second one allows a maximum of one child per parent node, the third a maximum of two children per parent node, etc) or deep learning based models (and assume number of parameters in the model reflects its complexity).

One key observation (and a potentially unappealing aspect) about this approach is: we aren't really splitting the data, we are generating new samples - hence the simulated data might lose nuance in the learning task. But the main reason I find the idea appealing and want to test it is because I feel the proposed approach can model a data generation process that isn't really a typical feature drift across organisation per se but might be common in a healthcare setting - where some hospitals handle a lot more complicated patients/procedures compared to the rest.

5.3 A setting where clients use different model architectures

How easy is it use the Flower framework when different clients have model architectures and the objective is to build a collaborative intelligence platform? I am aware that it is *possible* to do this with Flower. In fact, any federated setting can be implemented with Flower given at its core it is a communication/co-ordination framework. What I want to check is the ease of doing so.

Although using different model architectures isn't a very intuitive idea at first, there can be situations where it is sensible. One example could be privacy concerns in sharing models and hence opting to share updates via knowledge distillation process i.e., sharing the predictions of the local models on some public dataset. Frameworks like PATE [5] and tons of other literature on knowledge distillation try to do some variant of this, with different levels of success.

Fun question: Is the privacy risk of providing white-box access to a machine learning model the same as or more than providing black-box access to it?

Although the privacy risk from sharing models is a reasonable concern, my main motivation to think about clients with different model architecture was slightly different. Consider this scenario where different organisations have already built ML models to perform a particular task. They

have their own architectures and domain expertise embedded in these models. It is well-known that prediction scores from deep learning models don't really calibrate e.g., 40% of examples where the model assigns a probability of 0.4 to class A don't actually belong to class A. This can be a huge problem when the application requires an estimate of the uncertainty in the model's predictions. So, they intend tow work together for *calibrating* their models and offer a collaborative intelligence platform. How easy/hard is to build such a platform with Flower?

I think it would be a really cool capability if we can offer out-of-the box federated model calibration and uncertainty estimation tools. I am not a technical expert in this area and don't know how mature the field is but I am sure there is a lot of practical demand for such tools.

5.4 Simulate two different types of strategic clients and a platform coordinating their interactions

This is a purely speculative idea and something that gets me excited every single time. Consider the scenario of a platform such as Uber making its systems more *federated and intelligent*. Instead of it centralising the data about all its users and making decisions on behalf of both the drivers and riders, it offers strategic agents to both the parties. Only the agents have access to personal data and preferences of the users (e.g., how urgently a rider wants to go, their willingness to pay more or less money, etc). All interactions of the central platform with users (and their data) is only through these agents. What tools are currently available to distribute such strategic agents, if ever built? Now imagine if researchers inside Uber want to simulate the potential behavior and the evolutionary dynamics of their new algorithms for the strategic agents. What tools are available to simulate the interactions among such federated agents at large scale? Will this be an interesting direction for the Flower framework to evolve into?

Maybe ***train different*** could after all mean: *not training the same centralised intelligence differently but training a completely different type of intelligence - a truly federated intelligence.*

References

[1] Mikhail Khodak, Renbo Tu, Tian Li, Liam Li, Maria-Florina F Balcan, Virginia Smith, and Ameet Talwalkar. Federated hyperparameter tuning: Challenges, baselines, and connections to weight-sharing. *Advances in Neural Information Processing Systems*, 34:19184–19197, 2021.

[2] Leon Mächler, Ivan Ezhov, Florian Kofler, Suprosanna Shit, Johannes C Paetzold, Timo Loehr, Claus Zimmer, Benedikt Wiestler, and Bjoern H Menze. Fedcostwavg: A new averaging for better federated learning. In *International MICCAI Brainlesion Workshop*, pages 383–391. Springer, 2021.

[3] Leon Mächler, Ivan Ezhov, Suprosanna Shit, and Johannes C Paetzold. Fedpidavg: A pid controller inspired aggregation method for federated learning. *arXiv preprint arXiv:2304.12117*, 2023.

[4] Adnan Ben Mansour, Gaia Carenini, Alexandre Duplessis, and David Naccache. Fedcontrol: When control theory meets federated learning. *arXiv preprint arXiv:2205.14236*, 2022.

[5] Nicolas Papernot, Shuang Song, Ilya Mironov, Ananth Raghunathan, Kunal Talwar, and Úlfar Erlingsson. Scalable private learning with pate. *arXiv preprint arXiv:1802.08908*, 2018.

[6] Sarthak Pati, Ujjwal Baid, Maximilian Zenk, Brandon Edwards, Micah Sheller, G. Anthony Reina, Patrick Foley, Alexey Gruzdev, Jason Martin, Shadi Albarqouni, Yong Chen, Russell Taki Shinohara, Annika Reinke, David Zimmerer, John B. Freymann, Justin S. Kirby, Christos Davatzikos, Rivka R. Colen, Aikaterini Kotrotsou, Daniel Marcus, Mikhail Milchenko, Arash Nazer, Hassan Fathallah-Shaykh, Roland Wiest, Andras Jakab, Marc-Andre Weber, Abhishek Mahajan, Lena Maier-Hein, Jens Kleesiek, Bjoern Menze, Klaus Maier-Hein, and Spyridon Bakas. The federated tumor segmentation (fets) challenge, 2021.

A Potential issues with FedCostWAvg and FedPIDAvg

A.1 Privacy risk from the server monitoring loss related information

The intuition behind FedCostWAvg is quite appealing - if an update improved the performance of a local model by a lot, assign higher weight to it during parameter aggregation. But it also comes with additional privacy risk. A combination of loss information at each round along with the parameters could drastically increase the capability of the server to perform inference attacks on the clients. I haven't empirically tested this but I think it's possible to construct a very simple yet effective membership inference attack strategy for the server.

Given the server has access to M_i^j at client j after round i for multiple values of i and the loss improvement ratio k (in case of FedCostWAvg) at round i, it can map the pattern in values of k with the loss improvement ratios of a target point on models $M^j_{\{0,1,2,...i\}}$. The more similar the pattern, higher the chance that the target point is present at client j.

A.2 Instability in training with FedPIDAvg

Figure 2 shows that FedPIDAvg is a very unstable aggregation algorithm, especially during the overfitting phase i.e., when the accuracy on test data goes down. I suspect this is because of the loss differential term. Consider the following scenario with just two clients. Say the losses on local models moved as [0.5, 0.3, 0.21, 0.2, 0.25] at client 1 and as [0.6, 0.5, 0.2, 0.3, 0.55] at client 2 i.e., the performance improved until round 3 and got worse after that, especially for client 2. Now, let's calculate the weight assigned to these models during aggregation just based on the differential factor after round 3 and round 5. Before we get into the values, intuitively client2 should have higher weight in round 3 (improvement from 0.5 to 0.2 compared to 0.3 to 0.21)and client 1 should have higher weight in round 5 (less worse than client 2 because 0.2 to 0.25 compared to 0.3 to 0.55).

The weights from equation (5) after round 3 are: 9/39 for C1 and 30/39 for C2

After round 5 they are: (-0.05)/(-0.05 +(-0.25)) = 1/6 for C1 and (-0.25)/(-0.3) = 5/6 for C2

As expected, client 2 had a higher weight after round 3 but surprisingly it still had a higher weight even after round 5! This is because of taking the ratio of negative signed values, which flips the actual importance that's to be assigned to different clients.

We don't see any similar issues with FedCostWAvg precisely because there are no negative values in it (we do ratio of losses rather than differences). Notice that there is no way to fix this problem with FedPIDAvg by varying the relative weights of the differential and integral terms. To see this point, again consider the same loss values as above - client 2 will always have a larger sum of the loss values and hence higher weight. So, once the loss starts increasing at enough number of clients in any round, FedPIDAvg enters a positive feedback loop that keeps assigning higher weights to worse models, eventually collapsing to worst possible model, as seen in Figure 2b.