Personalized Federated Image classification using Weight Erosion

Damien Gengler, Martin Beaussart, Vincent Yuan

Supervisors: Felix Grimberg, Sai Praneeth Karimireddy, Mary-Anne Hartley

***Abstract*—Our goal in this project is to adapt Weight Erosion, an update aggregation scheme for personalized collaborative machine learning, to a neural network. Weight Erosion aggregates the data among different agents while ensuring privacy concerns - this scheme was developed for a medical concern, to create a performing machine learning model for infectious diseases such as Ebola. We then benchmark the WE scheme against two other common baselines for image classification on the MNIST dataset. We observe that WE outperforms both federated average and local training when the number of agents is quite high, thus when the number of samples per agents is quite low.**

I. Introduction

Federated learning (FL) techniques allow machine learning models to be trained on data spread across several agents, without exchanging the data between agents. The Weight Erosion (WE) scheme optimizes a personalized model for one agent using data from all agents. This differs from conventional FL methods (such as federated averaging) which trains a global model for all agents simultaneously. Indeed, FL techniques struggle when the distribution among the agents is non-IID [1]. The WE scheme solves this problem by customizing the contribution of each agent, depending on its similarities with the agent for whom we train the model.

One of the assets of aggregation schemes such as WE or FL is that the agents don't have to share sensitive data, but only their gradients. It allows the agents to take advantage of the other's agent data while ensuring that no agents share the data itself.

For example, during the 2014 Ebola Epidemic, data was fragmented across researchers and not shared, causing sometimes catastrophic delays in the public health response and diluting the statistical significance of findings. It took the researchers over 4 years to address the ethical concerns and centralize the data in the Infectious Disease Data Observatory [2]. Using FL in these cases could save some precious time as each hospital could train ML models with substantially more data than the ones of their own patients.

II. The Weight Erosion Scheme

The goal of Weight Erosion is to train a personalized model for a given requesting agent. The pseudo-code (algorithm 1), as well as all formulas for WE, are taken from the article by F. Grimberg et al. [3] and presented here without changes. This section gives a brief introduction to the scheme and the reader should refer to the paper for more details.

The key idea of WE is, at the round r and for the agent i, to attribute it a weight α_i^r based on the similarity between the gradient of $agent_i$ and the gradient of the requesting agent (Here, we consider $agent_0$ to be the requesting agent). The weights are then used to compute a personalized weighted average of the gradient vectors $\mathbf{g}_i$.

The weights are initialized at $\alpha_i^0 = 1$. At each round r, a gradient $\mathbf{g}_i$ is computed for all agents. We then compute for each $agent_i$ the relative distance $d_{i,0}^{rel}$ between its gradient $\mathbf{g}_i$ and $\mathbf{g}_0$,the gradient of the requesting agent (Equation 1).

Based on the relative distance, the previous round weight α_i^{r-1} and 2 hyperparameters, the distance penalty factor p_d and the size penalty factor p_s, the weight α_i^r of the agent is then computed (Equation 2).

We can note two things: First the agents' weights are always decreasing. Second the weight of the requesting agent is always 1. Hence, if we run the scheme for too many rounds, all weights will vanish except the weight of the requesting agent.

Finally the weight of the requesting agent is computed by taking the weighted average of the gradient of each agents (Equation 3).

The algorithm and the formulas are described below :

$$d_{i,0}^{rel} = \frac{\|\mathbf{g}_i - \mathbf{g}_0\|}{\|\mathbf{g}_0\|} \geq 0 \tag{1}$$

$$\alpha_i^r = \max\left\{0, \alpha_i^{r-1} - \left(1 + p_s \left\lfloor \frac{(r-1)b}{|\mathcal{S}_i|} \right\rfloor\right) p_d \cdot d_{i,0}^{rel}\right\} \tag{2}$$

$$\bar{\mathbf{g}} \leftarrow \frac{\sum_{i \in \mathcal{U}} \alpha_i^r \mathbf{g}_i}{\sum_{i \in \mathcal{U}} \alpha_i^r} \tag{3}$$

Grimberg et al. apply the new WE algorithm to a binary classification task using logistic regression with 7 features [3]. The first step of our project was to adapt WE to work on neural networks, so we could benchmark it on an image classification task. This is indeed an interesting adaptation as we know that neural networks are very powerful ML models. We will describe how we achieved this in next section.

III. Adaptation of the Weight Erosion Algorithm to Neural Networks

In this section, we talk about the adaptation of WE from an implementation point of view. We were given a federated learning framework using PyTorch [4]. Given the algorithm

Algorithm 1: WEIGHT EROSION

Data: A set of agents $i \in \mathcal{U}$, with associated data sets $\mathcal{S}_i$, a model class $\mathcal{M}$, and a gradient-based machine learning algorithm $\mathcal{A}$.
Result: A personalized machine learning model f.
Set a number of rounds r_{max}, a batch size b, a distance penalty factor p_d, and a size penalty factor p_s ;
Initialize $\alpha_i^0 \leftarrow 1 \ \forall i \in \mathcal{U}$;
Randomly initialize the model $f \in \mathcal{M}$;
for r *from* 1 *to* r_{max} **do**
  for $i \in \mathcal{U}$, *starting with* $i = 0$ **do**
    Select a batch of size b from $\mathcal{S}_i$ and compute a gradient $\mathbf{g}_i$;
    Compute the distance $d_{i,0}^{rel}$ (Equation 1) and α_i^r (Equation 2) ;
  end
  $\bar{\mathbf{g}} \leftarrow \frac{\sum_{i \in \mathcal{U}} \alpha_i^r \mathbf{g}_i}{\sum_{i \in \mathcal{U}} \alpha_i^r}$;
  Update the model f based on $\bar{\mathbf{g}}$;
end

for Weight Erosion, we could not directly implement it as such to our neural network. We encountered several challenges to adapt the scheme to our framework. These problems came from the difference in data structure between a neural network and a logistic regression model. Here is how we tackled those implementation problems.

The main challenge for the Pytorch implementation was the difference of how we represent the Gradient of the loss function w.r.t. the weights between a logistic regression model and a neural network. While it is pretty straightforward to define what is the gradient of a logistic regression at each step, there is not such a similar concept of what would be a unique Gradient defining a whole neural network. A neural network outputs a number of gradients equal to 2 times the number of layers minus one. Indeed, each layer except the first one has a gradient related to its weights, as well as a gradient related to the biases terms. So in the case of a neural network, instead of working with a single gradient, we must work with an array of gradients.

To solve this problem, we created a class to abstract these different gradients related to one neural network model: *GradientStocker*. This class stores the different gradients for one agent. Consider $agent_i$ with L hidden layers, the gradient is represented as:
$\mathbf{g}_i$ = [$\mathbf{g}_{i,1}$ ... $\mathbf{g}_{i,k}$ ... $\mathbf{g}_{i,2(L-1)}$] where $\mathbf{g}_{i,k}$ is the gradient of the loss function with respect to the weight or bias vector k. We then adapted the whole algorithm to apply it on a list of gradients. The gradient of agent i is used two times in the WE scheme: when computing $d_{i,0}^{rel}$ and when computing the weighted average of the gradients. We had to adapt our implementation to solve these problems.

In order to solve the first problem, we represented the 'total gradient' of one agent as being a tensor. This tensor is the concatenation of each layer's gradient: it is thus a vector. $d_{i,0}^{rel}$ is the normalized euclidean distance between two agents using this tensor.

For the second problem we need to compute the weighted average for each layer. Using the definition of gradient defined above, the final averaged gradient for the requesting agent is detailed in Equation 4.

$$\bar{\mathbf{g}} = [\frac{\sum_{i \in \mathcal{U}} \alpha_i^r \mathbf{g}_{i,1}}{\sum_{i \in \mathcal{U}} \alpha_i^r} ... \frac{\sum_{i \in \mathcal{U}} \alpha_i^r \mathbf{g}_{i,l}}{\sum_{i \in \mathcal{U}} \alpha_i^r} ... \frac{\sum_{i \in \mathcal{U}} \alpha_i^r \mathbf{g}_{i,2(L-1)}}{\sum_{i \in \mathcal{U}} \alpha_i^r}] \quad (4)$$

Finally, with these transformations in mind we can simply update the requesting model using the vector of gradients computed in Equation 4, performing a gradient descent for each layer.

The second challenge we encountered was an optimization problem. Indeed, as the neural network model is computationally expensive and knowing that we have to run a neural network for each agent, we had to optimize the scheme to not be overwhelmed by the computation.

At every round, we have to compute the gradient for each agent. To do so we use Gradient Descent techniques. The state of the art technique for large scale training in the mini-batch SGD. Instead of computing the gradient for the whole data we sample a batch and compute the gradient for that part of the dataset. The issue with this and distributed learning is that it creates a bottleneck on communications.

Instead, we choose to use Local SGD [5]. In this method, each agent does multiple iteration of minibatch SGD steps between each communication round. Each minibatch gradient is computed on a different model because the model is locally updated in-between. We decided to keep the minibatch size at 32, as suggested in this paper [6]. In order to get one gradient of the selected agent, we have to sum all of the gradients computed during the different iterations.

After resolving those challenges to adapt the Weight Erosion Scheme to a neural network, we wanted to benchmark this model to find out whether, and in which cases, WE outperforms other schemes on image classification tasks.

IV. BENCHMARKING

A. Aggregation Schemes

The goal of the Weight Erosion scheme is to provide a *personalized aggregation scheme* for a given agent using the help of other agents that have similar data, even though they might not be drawn from the same exact distribution. We decided to benchmark the WE scheme with two other baseline to compare their performances.

The first baseline is the *local training*. The local training will output a model which is relevant to the agent, using only the data available for this agent. This is the most straightforward approach. This model can perform poorly and have a high test loss compared to aggregation scheme. As local training doesn't take advantage of the others agents data, it will overfit more easily than the other schemes. There could be also a problem that the training set doesn't span well all of

the sample space (e.g. it contains only ones written like — but none written like 1). The local training will be able to achieve a high training accuracy, but it wouldn't generalize well to the test set and it would get a bad test accuracy.

This model will perform well if there is a lot of data available for the agent but we expect it to perform poorly in situations where the dataset will be small enough. This model is not going to receive any help from other agents. We include it in our benchmark to have a 'baseline model' as it is the 'classical' approach. If the WE scheme doesn't get a higher accuracy rate than the local training, then there's no point of using WE rather than training the data locally.

The second method is the *federated average* (Fed Avg). In this scheme, each agent shares its locally computed parameters. The results from each agents are then averaged and we use this non weighted average to update a global model. This scheme enables the different agents to share their locally computed models, without exchanging the data directly. This aggregation scheme is optimal when all data are drawn from the same distribution, but it performs quite poorly when the data of each agent are not independent and identically distributed [1].

B. Dataset

We benchmarked the WE scheme on the well known MNIST dataset, which consist of handwritten digits, from 0 to 9 [7]. The dataset is split into a training set and a testing set.

C. Hyperparameters and number of data points

To benchmark our three models we had to fix some hyperparameters, namely the learning rate γ and the number of communication rounds. Regarding WE specifically the distance penalty p_d and the size penalty p_s.

- The learning rate γ for all SGD updates was fixed at 0.1
- The number of communication rounds of the aggregation schemes (WE and FedAvg) was fixed at 30 with one epoch per round. for As we want to obtain the same behavior for all models, it means that for local training we did it with epoch equal to 30 as there is no communication rounds for this model.
- We did some tests to see how the accuracy changes with the speed at which the weights decline. We chose two distributions, one close to homogeneity and one far from homogeneity, and we analyzed the accuracy at each round for different distance penalties. We found out that we obtain better results when we keep for a long time weights greater to zero and just at the end of our number of rounds we get all our weight to zero (except for the one of the selected client). More precisely we tried to obtained the weight at zero around the round 20, we find this by fixing the distance penalty to 0.1/x where x is the number of clients.
- We made the number of clients vary, but the total amount of samples was kept the same and always divided equally across agents. As an example, with 10 agents, the roughly 60 000 samples of MNIST were divided yielding roughly 6000 samples per agent, and with 20 agents they each had roughly 3000 samples. The size penalty doesn't matter too much in our benchmark, as the size of the agent's dataset is almost always the same. That's why we arbitrarily decided to keep the size penalty at 2.

D. Distribution

Even though we benchmarked the different baselines with only one dataset (MNIST), we had multiples ways of partitioning the data set among the agents. A non-exhaustive list of the different points we could use to change the distribution is presented below:

- The total number of agents. We decided to compute our models for 10, 20, 50 and 100 agents
- The total amount of samples used. Here we use the MNIST dataset which contains roughly 60 000 images. We could vary the total number of samples to see how well the baselines perform in an environment where there is less information overall. In our case, we decided to keep the total dataset size the same, as we wanted to focus on different distributions and different number of agents.
- The label skew for each agent. For example a given agent get 20% of 1s, and 80% of 6s. This would also be interesting for WE as certain diseases might be very common in some places but rare elsewhere.

Here are the distributions we have benchmarked our baselines with.

DISTRIBUTION \ CLASS	0	1	2	3	4	5	6	7	8	9
A	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1
B	0	0	0	0	0,2	0,6	0,2	0	0	0
C	0,25	0,25	0,25	0,25	0	0	0	0	0	0
D	0	0	0	0,4	0,1	0	0,1	0,4	0	0
E	0	0	0	0,1	0,2	0,4	0,2	0,1	0	0
F	0	0	0,1	0,1	0,2	0,2	0,2	0,1	0,1	0
G	0,91	0,01	0,01	0,01	0,01	0,01	0,01	0,01	0,01	0,01

Figure 1. Repartition of digits between all agents for each distribution benchmarked

Table I presents the distribution of each digit's percentage for the first agent. Then, for each subsequent agent, we shift the distribution to the left, and put the distribution of the digit 0 as the digit 9.

Here's an example of how it works on Distribution B. Assume we have 10 agents.

$Agent_0$ will have 20% of 5s, 60% of 6s and 20% of 7s.

$Agent_1$ will have 20% of 6s, 60% of 7s and 20% of 8s.

And this shift of digits distributions goes on for all remaining agents. At the same time, we ensure that each datum point is assigned to exactly one agent.

Weight Erosion was developed to be used in heterogeneous environment so by varying the homogeneity of the distribution we can find the cases where it achieves better accuracy than Fed Avg or local training. Indeed, if the data distribution

among the agents is homogeneous (i.e. Distribution A), then we expect that FedAvg is the best baseline. On the opposite, if the data distribution among the agent is heterogeneous (i.e. $agent_0$ gets digits [0,1], $agent_1$ gets digits [2,3], etc.) then we expect that local training will provide the best test accuracy.

For each distribution, we compare the three algorithms in the cases where there are 10, 20, 50 or 100 agents. These values give a good idea of the evolution of the accuracy in several scenarios. We note that increasing the number of agents is equivalent to decreasing the number of samples per agent.

E. Result

We present a part of the results of our benchmarking in Table I below and the rest in Table II in the appendix.

Table I
RESULT TABLE – BEST ACCURACY – REDUCED VERSION

	Weight Erosion	***federated average***	***local training***
Distribution A			
10 agents	0.99	**0.991**	0.9719
20 agents	0.9819	**0.9869**	0.9568
50 agents	0.9759	**0.9769**	0.9277
100 agents	0.9649	**0.9709**	0.8896
Distribution C			
10 agents	**0.9981**	0.9894	0.9923
20 agents	**0.9952**	0.9875	0.9894
50 agents	**0.9952**	0.9798	0.9855
100 agents	**0.9933**	0.9701	0.9798
Distribution F			
10 agents	**0.9876**	0.9855	0.9731
20 agents	**0.9835**	0.9793	0.9607
50 agents	0.9669	**0.9762**	0.9287
100 agents	0.9545	**0.9597**	0.8936

V. DISCUSSION

A. Findings

We find that overall Weight Erosion shines when the number of agent is high. In these conditions, WE is quite often outperforming both Fed Avg and Local Training. Table II show the accuracy of the three methods over the number of rounds for some distribution with interesting properties. All other plots are available in the annex.

We show the graph for distribution B with 10 and 100 agents. From these two plots we can clearly see the evolution of the accuracy w.r.t. the number of agents. With 10 agents the algorithms accuracy evolve similarly and reach good test accuracy. Now for 100 agent, Local training and Federated Average's test accuracy decrease but Weight Erosion's accuracy stays quite high. These results generalize well for distributions B, C, D and E.

The plot for distributions F show us as expected that in situations where we tend toward a more homogeneous distribution, Fed Avg gives better test accuracy than Weight Erosion. This can be seen with distributions A and F.

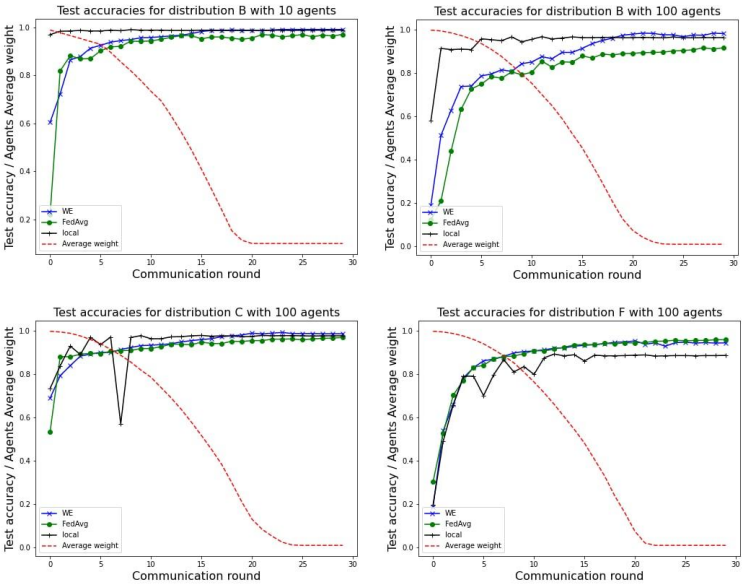

Figure 2. Evolution of the weight for WE and the accuracy of WE, Fed Avg and local training with different distribution and number of client

B. Limitations

- As we only benchmarked the different models on the MNIST dataset, we can't generalize our findings directly to other dataset.
- We kept the same amount of datapoints per agent for all distributions. This explain the fact that differences between method are small. Indeed this causes all methods to yield very high classification accuracies in all cases, and the accuracy difference is small between WE and FedAvg. This can be seen in Table II bottom left plot.

C. Possible Extensions

- To give results on a well know problem, we benchmarked on MNIST. This benchmark could be extended to other dataset such as EMNIST. The most interesting Dataset would be the suggested Ebola Dataset, since it is one of the use cases of WE.
- There are many different ways we could have played with the distribution, for example varying the total size of the dataset , or varying the dataset size among the agent instead of having all agents with the same number of samples.
- As explained, in section III we simply concatenate the gradients of different layer in the implementation. We could imagine giving a weight do different layer depending on their relevance with the task.

REFERENCES

[1] Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Keith Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. Advances and open problems in federated learning. *arXiv preprint arXiv:1912.04977*, 2019.

[2] B Goldacre, S Harrison, KR Mahtani, and C Heneghan. Who consultation on data and results sharing during public health emergencies. background briefing. sep 2015 centre for evidence-based medicine background briefing: Who consultation on data and results sharing during public health emergencies background briefing for who consultation on data and results sharing during public health emergencies. 2015.

[3] Felix Grimberg, Mary-Anne Hartley, Martin Jaggi, and Sai Praneeth Karimireddy. Weight erosion: An update aggregation scheme for personalized collaborative machine learning. In *Domain Adaptation and Representation Transfer, and Distributed and Collaborative Learning*, pages 160–169. Springer, 2020.

[4] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in neural information processing systems*, pages 8026–8037, 2019.

[5] Tao Lin, Sebastian U Stich, Kumar Kshitij Patel, and Martin Jaggi. Don't use large mini-batches, use local sgd. *arXiv preprint arXiv:1808.07217*, 2018.

[6] Dominic Masters and Carlo Luschi. Revisiting small batch training for deep neural networks. *arXiv preprint arXiv:1804.07612*, 2018.

[7] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010.

APPENDIX

Table II
RESULT TABLE ALL DATA – BEST ACCURACY

	Weight Erosion	***federated average***	***local training***
Distribution A			
10 agents	0.99	**0.991**	0.9719
20 agents	0.9819	**0.9869**	0.9568
50 agents	0.9759	**0.9769**	0.9277
100 agents	0.9649	**0.9709**	0.8896
Distribution B			
10 agents	**0.9913**	0.9707	0.9902
20 agents	**0.9935**	0.9707	0.9881
50 agents	**0.9859**	0.9544	0.9772
100 agents	**0.9848**	0.9176	0.9685
Distribution C			
10 agents	**0.9981**	0.9894	0.9923
20 agents	**0.9952**	0.9875	0.9894
50 agents	**0.9952**	0.9798	0.9855
100 agents	**0.9933**	0.9701	0.9798
Distribution D			
10 agents	**0.9901**	0.9752	0.9891
20 agents	**0.994**	0.9653	0.9831
50 agents	**0.9881**	0.9514	0.9722
100 agents	**0.9762**	0.9236	0.9653
Distribution E			
10 agents	**0.9915**	0.982	0.9873
20 agents	**0.9915**	0.9746	0.9789
50 agents	**0.9789**	0.9672	0.9567
100 agents	**0.9683**	0.9514	0.9376
Distribution F			
10 agents	**0.9876**	0.9855	0.9731
20 agents	**0.9835**	0.9793	0.9607
50 agents	0.9669	**0.9762**	0.9287
100 agents	0.9545	**0.9597**	0.8936
Distribution G			
10 agents	**0.9980**	0.9939	0.9929
20 agents	**0.9959**	0.9928	0.9898
50 agents	**0.9959**	0.9939	0.9898
100 agents	**0.9939**	0.9887	0.9857