DIFFERENTIAL PRIVACY AND BYZANTINE RESILIENCE IN FEDERATED LEARNING

Research project conducted by Morelle Dimitri in first year of master's degree in computer science

Supervised by Xu Chuan at the INRIA laboratory in the COATI team

Conducted from mid-October 2023 to mid-January 2024, submitted to Simonelli Anne-Laure and Di Giusto Cinzia

Côte d'Azur university student number : 21707461

Contents

ABSTRACT

Federated learning enables a large number of IoT devices (mobiles, sensors) cooperating to learn a global machine learning model while keeping the devices' data locally. For example, Google has applied FL in their application Gboard to predict the next word that the users would enter on their smartphones. During the training process, how to preserve the data privacy of the user and to prevent the failure of the learning due to malicious participants are two of the critical concerns in federated learning. For the privacy concern, differential private algorithms are introduced by injecting the noises to the transmitted messages. It ensures to a certain extent that even if the user changes just one training sample, the adversary should not observe a certain level of difference in the exchanged message and thus could not draw any conclusions. For the resilience of the system against the malicious participants, byzantine resilience algorithms are proposed to filter out the potential malicious updates from the users. However, recent research shows that a direct composition of these techniques makes the guarantees of the resulting algorithm depend unfavorably upon the number of parameters of the machine learning model, making the training of large models practically infeasible.

1 INTRODUCTION

The arrival of the digital era has led to a massive proliferation of data in diverse fields. This data represents a valuable resource for training machine learning (ML) models capable of providing relevant predictions. However, the centralisation of this data raises major concerns in terms of confidentiality and security.

As phones and tablets are the most widely used computing devices in the population, combined with the fact that they contain a huge amount of data (including GPS, microphones and cameras), they are very valuable sources of data, but also run the risk to storing it in a centralized location. For example, a model to aid typing on a mobile keyboard is better served by training data typed in mobile apps rather than from scanned books or transcribed utterances. However, language data can be uniquely privacy sensitive. In the case of text typed on a mobile phone, this sensitive information might include passwords, text messages, and search queries.

Federated learning (FL) emerged in 2016, following an article [1] published by Google researchers, as an innovative solution to address these issues. In essence, it enables collaborative training of ML models on decentralised data, without the

data leaving its original sources. This paradigm proposes to preserve the privacy of the data while enabling the creation of robust and generalisable models.
Differential privacy (DP) ensures that the results of ML do not reveal any specific information about an individual. And that this protection or even the absence of an individual in a data set will have only a negligible impact on the final results. The DP mechanism introduces random noise into the analysis or learning process, making it difficult, if not impossible, to determine whether a particular observation is the contribution of a specific individual. By adding this noise in a controlled manner, DP allows sensitive information to be hidden while preserving the overall usefulness of the results.
In addition to noise, an important part of DP is layer clipping. This technique is used in FL to reduce the potential problems associated with gradient explosions during model training. This occurs when the gradients of the model parameters become extremely large, which can lead to instabilities in the optimisation process. In practice, layer clipping involves limiting the magnitude of the gradients for each model layer during the training phase. More precisely, a threshold value is defined, and if the norm of the gradient of a layer exceeds this threshold, all the gradients of the layer are proportionally reduced to respect the threshold.
This technique helps to stabilise the model training by preventing the gradients from becoming excessively large. When gradients are too large, this can lead to significant parameter updates, which can compromise model convergence and lead to divergences.
In this project, by using a FL approach along with DP, we will be able to carry out several experiments to analyse the best parameters for ensuring a robust model while preserving the confidentiality of customers' data as effectively as possible.

2 STATE OF THE ART

At the moment, FL is based on the Federated Averaging algorithm (FedAvg) proposed in the paper [1]. This algorithm initializes the parameters of the global model and organizes the training process into rounds and repeating for each subsequent round. In each round, the server selects a random subset of clients from the set of available clients. Each selected client updates its local model in parallel with the other clients using its own dataset. This is done by diving the local data into batches of a certain size and running several local epochs. During each epoch, the local model is updated according to the gradient of the model with

respect to the batch data. Once all clients have performed their local updates, they send back the local updates to the server. The server then updates the global model with respect to an aggregation rule. This allows a global model to be trained without clients' data leaving their devices, which preserves data confidentiality.
However, even though the data is kept locally by the devices, it has been shown that an honest-but-curious server can still reconstruct data samples [2,3], sensitive attributes [4,5] and the local model [6] of a targeted device. In addition, the server can conduct membership inference attacks [7] to identify whether a data sample is involved in the training or source inference attacks to determine which device stores a given data sample [8]. Differentially private algorithms [9] have been proposed to tackle the challenges of protecting user privacy.
Differential private scheme guarantees that when one of the users is removed from the training, the distribution of the observations is still close to the one with all the users. The distance of the distributions is measured by the privacy budget ε, i.e., smaller ε presents higher privacy protection.
These algorithms work by adding noise to transmitted messages, ensuring that minor alterations in a user's training dataset will not be discernible to potential adversaries [10,11]. In particular, [10] proposes an approach to train large recurrent language models while preserving user-level DP in FL setting. It demonstrates that it is possible to protect the privacy of individual users' data with only a minimal impact on predictive accuracy. The authors achieve this by incorporating user-level privacy protection into the FedAvg, which facilitates large-step updates from user-level data. With a sufficiently large dataset, privacy preservation results in increased computational costs rather than a significant loss in utility. Using an estimator with limited sensitivity, [10] Gaussian noise proportional to this sensitivity can be added to guarantee confidentiality. Rather than using a classical (ε, δ)-DP approach, using the "Moments Accountant" of Abadi et al. (2016a) allows for more accurate confidentiality guarantees by relating the total confidentiality cost to T-steps of a sampled Gaussian mechanism with noise $N(0, \sigma 2)$ for $\sigma = z \cdot S$, where z is a parameter, S is the sensitivity of the query, and each row is selected with probability q. Given a $\delta > 0$, the accountant gives an ε for which this mechanism satisfies (ε, δ)-DP.

3 METHODS

1 - Environment

In this context, I explored and adopted the Flower framework, a platform dedicated to FL. Integrating Flower into the research project proved essential for the practical implementation of the FedAvg algorithm.

To be able to implement the DP algorithm [10], I used an open-source library Opacus that enables training Pytorch models with DP. It supports training with minimal code changes required on the client, has little impact on training performance, and allows the client to online track the privacy budget expended at any given moment.
I succeed to implement the private FL framework [10] by integrating Opacus into Flower. The dataset used for the training is CIFAR-10, it consists of 60,000 32x32 color images in 10 classes, with 6,000 images per class. There are 50,000 training images and 10,000 test images. We choose the convolution neural network (CNN) architecture with 2 convolutional layers and 3 linear layers for the training model. We consider first two client's scenarios. The dataset is randomly split over two clients.

The choice of the epsilon (privacy budget) value is crucial: a smaller value means stronger privacy protection whileas greater disruption of the model performance. A higher epsilon value allows more accurate learning but may reduce the level of privacy. The challenge is to find an appropriate balance between privacy and model accuracy by adjusting the epsilon value to the specific needs of the application. This is what we checked on 10 runs of the algorithm, and the results were as expected shown in Figure 1.

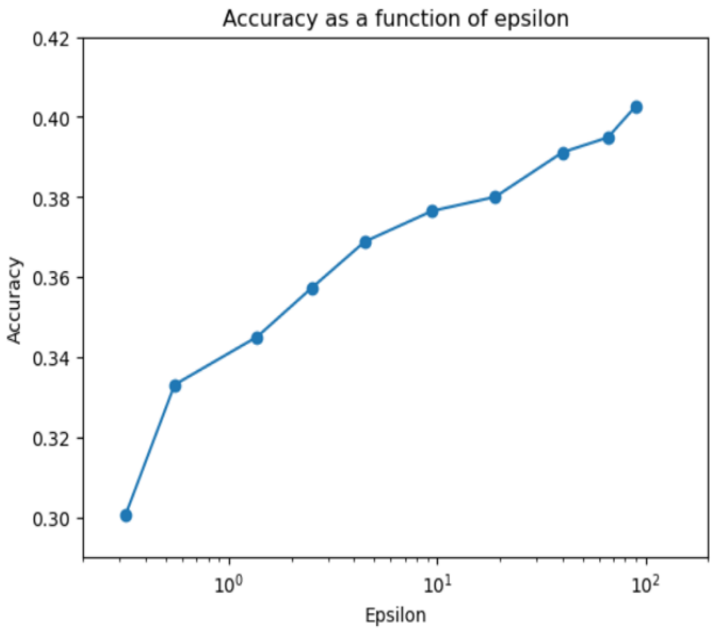

Figure 1 : The privacy budget ε (on a logarithmic scale) and the associated test accuracy.

2 - Clipping strategies

Before clipping the different layers, we can also look at the proportion of gradients in each of them. The magnitudes of the gradients generally decrease as you progress through the layers. This may indicate that the deeper layers of the network require finer adjustments and may have already converged further. On the other hand, if the gradients are very high in certain layers, this could indicate a potential need for clipping in those specific layers (more than others) to avoid excessive updates. As can be seen in Figure 2 below, the first layer of convolutional neural networks has higher gradient values, as opposed to the last layer of linear networks. We can then use these results to adjust our per-layer clipping method.

We will mainly be comparing 3 clipping methods. First is flat clipping, which consists of applying a uniform limitation to all gradients, regardless of their value; it applies globally to the gradient norm. If the norm of the gradient exceeds a certain predefined threshold, then all the gradients are reduced proportionally so that the final norm is equal to this threshold. Formally, based on [10] suppose each user update Δk contains m vectors $\Delta k = (\Delta k(1), \ldots, \Delta k(m))$ we clip the concatenation of all the layers as $\Delta' k = \prod(\Delta k, S)$, with S as a threshold.

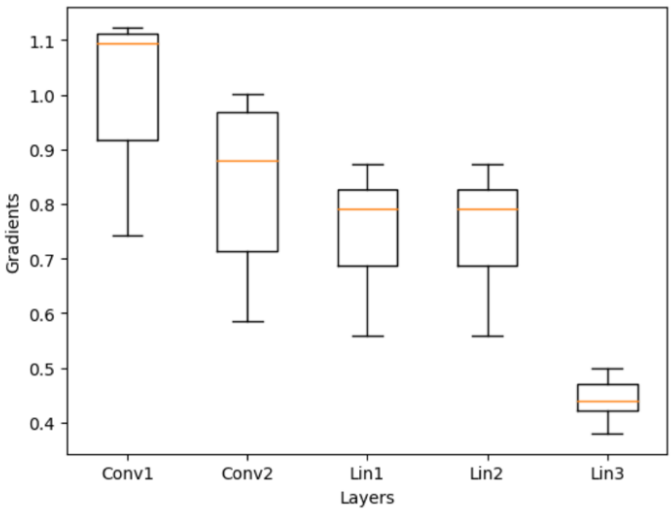

Figure 2 : Boxplot of gradient magnitudes per layer of the neural network.

The second and third clipping methods are both the per-layer clipping, this procedure involves defining a specific clipping threshold for each layer of the network. During back-propagation, if the norm of the gradients of the layer exceeds its clipping threshold, all the gradients of this layer are reduced proportionally so that the final norm is equal to this threshold.
We will first do it uniformly, still based on [10], given a per-layer clipping parameter S_j for each layer, we set $\Delta' k(j) = \prod(\Delta k(j), S_j)$. Let S = $\sqrt{\sum_{j=1}^{m} S^2{}_j}$ with $S_j = \frac{S}{\sqrt{m}}$ for all j.

On the other hand, to not do it uniformly we will adjust the parameter S_j proportionnaly to the maximum value of the gradients M_j .

We will therefore use $S_j = S_{j+1} \cdot \frac{M_j}{M_{j+1}}$.

The advantage of not clipping it uniformly is that it allows finer adaptation to the different learning dynamics of each layer, so we may expect a better accuracy for the same privacy budget as the uniform per-layer clipping. More simply, we use several thresholds rather than a single one as in the uniform or flat clipping method.

Neural layers have two fundamental components: the weight, which is a parameter learned during the training phase, actually each connection between neurons is associated with a weight, these weights determine the impact of the input of a neuron on the output of neurons connected in the following layer, and they are adjusted during training in order to optimise the network's performance on a specific task. And the bias, which is used to adjust the overall behaviour of the output layer. For practical reasons, we will only use the values of the weights to carry out the computations and the plots.

4 EVALUATIONS

In this section, we will evaluate these 3 clipping methods with only varying the noise and the privacy budget. Training will take place over 5 rounds and we will capture automaticly the average accuracy for these 5 rounds. In the graph below, each point corresponds to a training run, each with a different noise and epsilon. We vary the noise between 2 and 0.22 over 13 executions.

Regarding the structure of the model, during forward propagation, images pass through convolution layers, followed by group normalisations and pooling operations. Next, the resulting activations are flattened for use by fully connected layers, each applying a tanh activation function. The final layer produces class scores without an activation function. During training, a loss function measures the difference between the predicted scores and the actual labels, and a gradient descent adjusts the weights to minimise this loss. The predicted class is determined by taking the index of the highest score during prediction.

```python
class Net(nn.Module):
    def __init__(self) -> None:
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.gn1 = nn.GroupNorm(int(6 / 3), 6)
        self.pool = nn.AvgPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.gn2 = nn.GroupNorm(int(16 / 4), 16)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x: torch.Tensor) -> torch.Tensor:
        x = self.pool(torch.tanh(self.conv1(x)))
        x = self.gn1(x)
        x = self.pool(torch.tanh(self.conv2(x)))
        x = self.gn2(x)
        x = x.view(-1, 16 * 5 * 5)
        x = torch.tanh(self.fc1(x))
        x = torch.tanh(self.fc2(x))
        x = self.fc3(x)
        return x
```

As we can see here, the image classification model actually uses 7 layers of neural networks, and to facilitate the analysis while retaining the fundamental aspects of information processing by the neural network, we take for the analysis, as said previously, only the convolutional and linear layers.

On this figure 3 we are not using the logarithmic scale as the figure 1. The difference in accuracy between the three clipping methods varies for the last execution between 0.37 for the uniform per layer clipping method, 0.38 for the per layer adaptative clipping method and almost 0.40 for the flat clipping method. The variation of the noise between 2 (first execution) and 0.22 (last execution) causes the privacy budget epsilon to vary from almost 0 to 90.

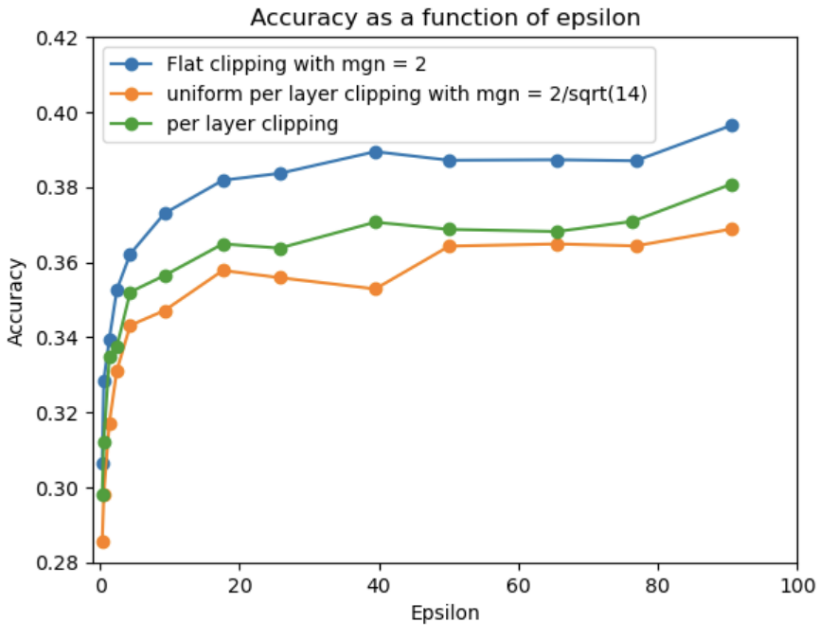

Figure 3 : The effect of different level of noise σ varying the privacy budget ε for the three clipping methods with S = 2

All interpretations of this figure 3 will be indicated in the next paragraph "Discussion".

However, here is the part of a script with all the arguments used during execution.

```
# Loading script arguments
NBCLIENTS="${1:-2}" # Nb of clients launched by the script (defaults to 2)
NBMINCLIENTS="${2:-2}" # Nb min of clients before launching round (defaults to 2)
NBFITCLIENTS="${3:-2}" # Nb of clients sampled for the round (defaults to 2)
NBROUNDS="${4:-5}" # Nb of rounds (defaults to 3)
VBATCHSIZE="${5:-256}" # Virtual batch size (defaults to 256)
BATCHSIZE="${6:-256}" # Batch size (vbatchsize%batchsize=0) (defaults to 256)
LR="${7:-0.01}" # Learning rate (defaults to 0.01)
NM="${8:-0.22}" # Noise multiplier for the Privacy Engine (defaults to 1.0)
MGN="${9:-1.2 1.1 1 0.5 0.3}" # Max grad norm for the Privacy Engine (defaults to 1.2)
EPS="${10:-100.0}" # Target epsilon for the privacy budget (defaults to 3.0)
```

We can see that the learning rate value is 0.01 and we never changed it in all the executions, it might be modified to have an optimal convergence, to increase the accuracy based on the model, preventing over-fitting... But to find the good learning rate value for all that points would be as complex as analysing these different clipping methods.

5 DISCUSSION

In this study, we explored the effectiveness of three different clipping methods in the context of federated learning, evaluating their performance as a function of the privacy budget defined by the epsilon parameter. The methods evaluated were flat clipping, per layer adaptive clipping and per layer uniform clipping.

The flat clipping method showed the best overall performance. Flat clipping applies a uniform threshold throughout the model. This approach may be more efficient as it simplifies the clipping process by treating all layers equally, reducing complexity and potential errors. However, this method may not be optimal for models where some layers require finer or different processing.

Per layer adaptive clipping, which ranks second in terms of performance, allows greater flexibility. By adjusting the clipping for each layer adaptively, this method can theoretically provide a balance between protecting privacy and preserving useful information for each specific layer. However, this can introduce additional complexity in setting clipping thresholds, requiring careful attention to avoid over- or under-adjustment.

Finally, the per layer uniform clipping method was the least effective in our study. By applying a uniform but specific threshold to each layer, it seeks to find a middle ground between flat clipping and per layer adaptive clipping. However, the challenge with this approach lies in determining an appropriate threshold for each layer without the flexibility offered by adaptation.

When considering the epsilon privacy budget, it is crucial to note that the effectiveness of these clipping methods can vary significantly. A lower epsilon promotes better privacy but can reduce the accuracy of the model. The performance of flat clipping suggests that, for some models and datasets, a more simplified approach may be sufficient to achieve a balance between privacy and accuracy.

So the choice of clipping method in FL is highly dependent on the specific nature of the model and data, as well as privacy requirements. Our results highlight the importance of carefully evaluating different clipping strategies to optimise both model performance and user privacy.

6 CONCLUSION

The study conducted on FL, focusing on the evaluation of different clipping methods according to the epsilon confidentiality budget, reveals significant insights for the data science community. Our results clearly demonstrate that the clipping method has a considerable impact on model performance, while respecting confidentiality constraints.

This research contributes to the understanding of the delicate balance between privacy preservation and model efficiency in FL. It paves the way for future studies to further explore how different clipping strategies can be optimised for specific applications, while respecting privacy constraints.

In fact, for the entire structure of the project, i.e. the dataset, the rather simple architecture of the model, a fairly small number of rounds and the separation of data into just 2 clients, flat clipping proved more effective.

We could therefore imagine carrying out future studies on much more complex models and other datasets to analyse whether the clipping per layer method might be more effective. And also play with other settings as the learning rate, as we said before.

In conclusion, FL, as a rapidly expanding field, presents unique challenges in terms of privacy protection and model performance. The results of this study underline the importance of a methodical and tailored approach to addressing these challenges, with a view to promoting significant advances in the responsible and efficient exploitation of distributed data.

BIBLIOGRAPHY

[1] McMahan et al, Communication-Efficient Learning of Deep Networks from Decentralized Data, AISTATS 2017
[2] Geiping et al, Inverting gradients - how easy is it to break privacy in federated learning?, NeurIPS 2020
[3] Yin et al, See through gradients: Image batch recovery via gradinversion, CVPR 2021
[4] Lyu et al, A novel attribute reconstruction attack in federated learning, FTL-IJCAI 2021
[5] Driouich et al, A novel model-based attribute inference attack in federated learning, FL-NeurIPS22, 2022.
[6] Xu et al, What else is leaked when eavesdropping Federated Learning?, PPML-CCS, 2021
[7] Zari et al, Efficient Passive Membership Inference Attack in Federated Learning, PriML-NeurIPS workshop, 2022
[8] Hu et al, Souce inference attacks in federated learning, ICDM 2021
[9] Dwork and Roth, A. The algorithmic foundations of differential privacy. Now Publishers Inc., 2013.
[10] McMahan et al, Learning differentially private recurrent language model, ICLR 2018
[11] Bellet et al, Personalized and Private Peer-to-Peer Machine Learning, AISTATS 2018