Simulation on Differentially Private Federated Meta-learning Systems

Authors: Ning Wang', Yang Xiao†, Yimin Chen‡, Ning Zhang*, Wenjing Lou', and Y. Thomas Hou'

Virginia Tech' 
University of Kentucky† 
University of Massachusetts Lowell‡ 
Washington University in St. Louis*

Our ACSAC paper is entitled “Squeezing More Utility via Adaptive Clipping on Differentially Private Gradients in Federated Meta-Learning”.

Background: Federated Learning (FL)

- Key Characteristic: not requiring data sharing.
- Goal of FL: enable a central server to train a global model by aggregating model parameters from distributed intelligent end devices.
Research Problem and Goals
Privacy Attack against FL

- FL cannot guarantee the privacy of training data.
- State-of-the-art Inference Attack
  - Model Inversion Attack [3]
  - Membership inference attack [1,2]
  - Attribute Inference Attack [4]

Problem and Goal of this Paper

- **Problem**
  - FL can protect data privacy to some extend.
  - Attackers are still capable to infer training data while knowing the model parameters.
  - Differential Privacy (DP) is a tool for privacy protection, but it harms the accuracy a lot.
  - We consider a scenario that local data is small.

- **Goal**
  - Provide rigorous privacy guarantee for users by incorporate DP.
  - Maintain a good trade-off between privacy and accuracy.
Differentially private federated Learning

- Problem: low accuracy

Continuous Training does help.
But it requires Enough Data & Training Power/Time
Differentially private federated Meta-Learning

- Federated Learning $\rightarrow$ **Federated Meta-Learning**
  - Deal with few-shot problem.
  - Fast customization.

![Diagram of federated Meta-Learning process]

- Mobile client
- Support set
- Query set
- Task 1
- Task 2
- Trained model
- Aggregator
- Meta Model
  (fast adapted to different domains)
Proposed Workflow, Experiment and Evaluation
● Adding noise

- The noise should be proportional to the largest gradient.
- To avoid too large noise, we should clip the gradient.
Our Proposal

- **Adaptive Clipping**
  - Naïve constant clipping maintain a fixed clipping threshold $C$. The noise will be: $k \times C$.
  - Adaptive clipping: change the threshold $C$ adaptively.

The gradients will decrease during the course of training.

We can change the threshold $C$ according to the gradient change.
Differentially private Meta-learning

- The history of *Differentially Private* version gradients guides the current clipping.
Two Algorithms

- Two threat models
  - DP-AGR for threat model 1 where server is trusted, clients are honest-but-curious
  - DP-AGRLR for threat model 2 where the server is not trusted, and clients are honest-but-curious

```plaintext
Algorithm 3: DP-AGRLR (Client Side)

Input: Current global model Θ, local data D, DP parameter (ε₀, δ₀), C₀, z₀
Output: gradient g

1. Function g = Base-Model-Train(Θ, D, D[q]):
2. Initialize base-model: θ ← Θ;
3. Split local data D[s], D[q] ← D;
4. z₀ ← compute_noise(ε₀, δ₀, *args)
5. for (xᵢ, yᵢ) ∈ D[s] do
6.   record-level gradient: ̂gᵢ ← ∇θ L(θ, xᵢ);
7.   clip gradient: ̃gᵢ ← ̂gᵢ * min(1, C₀∥̂gᵢ∥);
8.   ̃g ← 1/|D[s]| (∑ₖ ̃gᵦ + N(0, (z₀C₀)²I));
9. update base-model: θ ← θ − η₁ ̃g;
10. for (xᵢ, yᵢ) ∈ D[q] do
11.   record-level gradient: ̂gᵢ ← ∇θ L(θ, xᵢ);
12.   clip gradient: ̃gᵢ ← ̂gᵢ * min(1, C₀∥̂gᵢ∥);
13. g ← 1/|D[q]| (∑ₖ ̃gᵦ + N(0, (z₀C₀)²I)).
```
Experimental Setting

- **Settings:**
  - Image Datasets: Omniglot, CIFAR-100, mini-ImageNet
  - Client Number: 400,000
  - Clients in each learning round: 1500
  - Each client has 30 data records.
  - Meta-learning algorithm: MAML.
    - [https://github.com/AntreasAntoniou/HowToTrainYourMAMLPytorch](https://github.com/AntreasAntoniou/HowToTrainYourMAMLPytorch)

- **Code:**
  - Code Evaluated
Datasets (1/3)

- Omniglot Dataset [https://github.com/brendenlake/omniglot](https://github.com/brendenlake/omniglot)
- 1623 characters
- Each has 20 examples
Datasets (2/3)

- 100 classes
- Each has 600 examples
Datasets (3/3)

- CIFAR-FS Dataset [https://github.com/bertinetto/r2d2](https://github.com/bertinetto/r2d2)
- 100 classes
- Each has 600 examples
Data Split

- Use CIFAR dataset as example: 100 classes, each has 600 examples.
- A general image classification
  - Training: 100 classes, each has 500 examples
  - Testing: 100 classes, each has 100 examples
- Meta-learning
  - Training: 80 classes, each has 600 examples
  - Testing: 20 classes, each has 600 examples
N-Way K-shot Task

- $N$ denotes the number of classes.
- $K$ represents the number of data records in each class.

- N-way K-shot meta-learning
  - **Meta-Training:**
    - Pick $N$ classes, pick $K$ records for each of the $N$ classes, **learn a base model.**
    - Pick other records in the $N$ classes to calculate gradients on the learned **base model.**
    - Use gradients to update **meta** model.
  - **Meta-Testing**
    - Pick $N$ **unseen** classes. pick $K$ records for each of the $N$ classes. **Continuously train the meta-model** using these data.
    - Pick other records in the $N$ classes to evaluate accuracy.
N-Way K-shot Task

- Visualization of 2-way 3-shot

Train a base model

Update meta-model.

Calculate gradients with new data and base model.

Continue to train meta-model

Test accuracy of the trained meta-model.

training

airplane
automobile
bird
cat
deer
dog
frog
horse

testing

ship
truck
Simulate Clients

- Sampling data.
- 400,000 clients

Client 1
- airplane
- automobile
- bird
- cat
- deer
- dog
- frog
- horse

Client 2
- training

Client 3
- testing

ship
truck
Deep learning Environment

- **Hardware**: a server equipped with a 3.3 GHz Intel Core i9-9820X CPU, three GeForce RTX 2080 Ti GPUs
- **Operating system**: Ubuntu 18.04.3 LTS (a different version of Ubuntu is okay if it supports the Pytorch deep learning framework)
- **Deep learning framework**: Pytorch 1.4.0
- **Programming Language**: Python 3.6.10
Set up GPU for Deep Learning

● Goal
  ■ Enable Deep Learning libraries (e.g., Pytorch) talk to GPU.

● Setting Up Steps:
  ■ NVIDIA Driver installation
  ■ CUDA installation
  ■ and CUDNN installation

● Setting up guidelines are available on a blog
  https://towardsdatascience.com/deep-learning-gpu-installation-on-ubuntu-18-4-9b12230a1d31
Adaptive Clipping Percentile $k$

- The adaptive clipping threshold at time step $t + 1$ is computed with a sequence of differentially private version of gradients before $t + 1$ (i.e., $\tilde{g}_{t-W+1}, \tilde{g}_{t-W+2}, \ldots, \tilde{g}_t$) by

$$C_{t+1} := f(\{\tilde{g}_{t-W+1}, \ldots, \tilde{g}_t\}, k)$$

All other settings are the same, only change the clipping method.
Noise Level $z$

- Small Noise will consume the privacy budget quickly, so learning iterations will be limited.
- Larger Noise will cover useful gradients.
- Explore Trade-off.

Noise=1 get the best accuracy. It indicates learning iteration is not a key limitation on the used dataset.
Client Sampling Number $L$

- With small sampling number, noise may cover the gradients.
- With large sampling number, algorithm will reach the privacy leakage threshold quickly, so learning iterations will be limited.
- Explore Trade-off

L=1600 get the best accuracy.
Explore Client Sampling Number $L$ and Noise $z$

Together

- **When $L$ is small,**
  - The smaller noise get better accuracy.
  - Because larger noise may cover the gradients.

- **When $L$ is large enough,**
  - The smaller noise get extremely low accuracy.
  - Because the combination of large $L$ and small $z$ will reach the privacy leakage threshold quickly, so learning iterations will be limited.
System Evaluations over Baselines

- privacy budget
  - DP-AGR achieves $(1.5, 10^{-6})$-DP;
  - DP-AGRLR achieves $(2.5, 10^{-5})$-DP for record-level privacy
  - Baseline achieves $(9.5, 10^{-3})$-DP

- Accuracy

<table>
<thead>
<tr>
<th>Dataset</th>
<th>N-way K-shot</th>
<th>Random initial</th>
<th>Non-private</th>
<th>Private Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DP-AGR</td>
</tr>
<tr>
<td>Omniglot</td>
<td>5-way 1-shot</td>
<td>49.2</td>
<td>99.4</td>
<td>93.9</td>
</tr>
<tr>
<td></td>
<td>5-way 5-shot</td>
<td>61.0</td>
<td>99.8</td>
<td>96.8</td>
</tr>
<tr>
<td>CIFAR-FS</td>
<td>5-way 1-shot</td>
<td>33.8</td>
<td>61.0</td>
<td>47.1</td>
</tr>
<tr>
<td></td>
<td>5-way 5-shot</td>
<td>45.4</td>
<td>78.6</td>
<td>58.2</td>
</tr>
<tr>
<td>Mini-ImageNet</td>
<td>5-way 1-shot</td>
<td>23.3</td>
<td>51.7</td>
<td>37.3</td>
</tr>
<tr>
<td></td>
<td>5-way 5-shot</td>
<td>24.2</td>
<td>65.3</td>
<td>48.8</td>
</tr>
</tbody>
</table>
Hardware Evaluation Results

- We are running the code on a server equipped with a 3.3 GHz Intel Core i9-9820X CPU, and a GeForce RTX 2080 Ti GPU. The running time of DPAGR

<table>
<thead>
<tr>
<th>Dataset</th>
<th>MAML</th>
<th>DP-AGR</th>
<th>DP-AGRLR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Omniglot</td>
<td>39.9ms</td>
<td>54.7ms</td>
<td>0.52s</td>
</tr>
<tr>
<td>CIFAR-F</td>
<td>68.7ms</td>
<td>81.3ms</td>
<td>1.06s</td>
</tr>
<tr>
<td>Mini-ImageNet</td>
<td>112.3ms</td>
<td>102.7ms</td>
<td>1.17s</td>
</tr>
</tbody>
</table>

- DP-AGR achieves comparable computational performance with the original non-private MAML algorithm.
- DP-AGRLR is more time-consuming due to the need for computing per-record gradients.
Discussion & Meta Questions
Base Code

- A baseline Meta-learning algorithm: MAML. ([https://github.com/AntreasAntoniou/HowToTrainYourMAMLPytorch](https://github.com/AntreasAntoniou/HowToTrainYourMAMLPytorch)). It’s a centralized non-private meta-learning algorithm.
- The privacy evaluation library, moments accountant: [https://github.com/tensorflow/privacy](https://github.com/tensorflow/privacy)
Reproduced Baselines

- We reproduced the results of Base code as one baseline.
  - MAML: [https://github.com/AntreasAntoniou/HowToTrainYourMAMLPytorch](https://github.com/AntreasAntoniou/HowToTrainYourMAMLPytorch).
  - Their results are reproduced.
- For the other baseline GBML [1], code is not published.
  - We reimplemented their methods.
  - Our produced results were different from their reported results, some with higher accuracy while some with lower accuracy.
  - We used two common datasets with GBML. For the two datasets, I ended up copying their results published in their paper.
  - For another dataset, we report the results evaluated by our reimplemention.

Lessons Learned

- Guidelines for \( k, z, \) and \( L \)
  - First, we recommend to start from a small noise multiplier \( z \) (e.g., 1) and increase \( z \) only when you can not guarantee convergence before using up the privacy budget.
  - Second, we recommend starting with a relatively large \( L \) especially when \( z \) is large.
  - We can decrease \( L \) only when you can not guarantee convergence before using up the privacy budget.
  - Compared with the non-private training, we need apply a larger learning rate since the training rounds are limited because of privacy concerns.
  - Finally, as privacy parameter \( \epsilon \) is only determined by \( z \) and \( L \), we can adjust other parameters, such as \( k \), to boost the model accuracy.
Slow Training Problem

- Problem
  - The training time of DP-AGRLR for 400,000 tasks is over 30h.

- Reason
  - Per-record gradient calculation is time-consuming, and it’s an open problem.
Slow Training Problem

- **How to deal?**
  - GPU speeds up gradient calculation for **batched** data.
  - If we still load a batch of data, but calculate gradient one record by another. There will be much I/O between GPU and CPU.
  - Loading one record a time.

- **What did we achieve?**
  - Training time from 30h to 6h.
Wrap up discussion
Summary

- Differentially private federated meta-learning architecture.
  - Parameter tuning is time-consuming
  - We should reason the tuning directions beforehand but not tune randomly.
  - Good trade-off can be found.
- Understand how GPU speeds up to avoid wrong configurations.
Future Direction

- Further improve the time efficiency by using state-of-the-art single-record gradient acceleration techniques [1].
- Differential Privacy is represented by two parameters $(\varepsilon, \delta)$.
  - It is not straightforward to understanding how good the privacy is.
  - We plan to implement privacy attack.
    - Membership inference attack [2,3]
    - Model Inversion Attack [4]
    - Attribute Inference Attack [5]
  - Attack on differentially private model and non-private model.
  - The attack success rate can be an indicator for the privacy protection level.

Thank You!
Q&A