Project: Attacking Gradients: reconstructing training data

Presenter : Ziyu Li,

March 16, 2024

IP PARIS

Table of Contents

Multi-node Training

A brief overview of multi-node training and potential risk

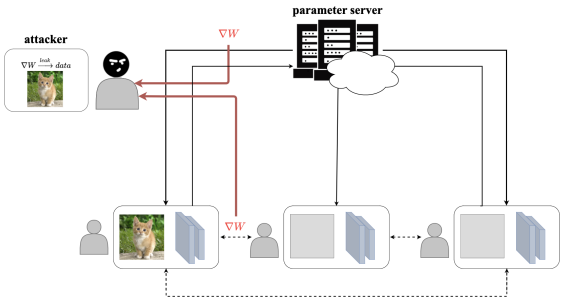


Figure: Multi-node training framework.

- Independent **Nodes**
- Local **Dataset**
- Shared **Information**

Objective of our work

Objective

Objective :

- Implement a **Deep Leakage Gradient** attack and its **improved method**.
- Evaluate the sensibility of **weight initialization** to **reconstruction quality**.

Table of Contents

Deep Leakage from Gradient

Principle

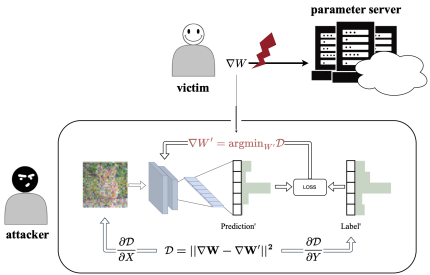


Figure: The gradient transmitted by the victim to the server is intercepted by the attacker.

Dummy Gradient:

$$\nabla W' = \frac{\partial \mathcal{L}(F(x', W), y')}{\partial W}$$

Dummy Data & Dummy Label:

$$x'^*, y'^* = \underset{x', y'}{\text{argmin}}(\text{loss}) = \underset{x', y'}{\text{argmin}} \left\| \nabla W' - \nabla W \right\|^2$$

$$= \underset{x', y'}{\text{argmin}} \left\| \frac{\partial \mathcal{L}(F(x', W), y')}{\partial W} - \nabla W \right\|^2$$

Deep Leakage from Gradient

From Mean Square Error to Cosine Dissimilarity

$$\text{loss} = \left\| \nabla W' - \nabla W \right\|^2$$

Mean Square Error

$$\text{loss} = 1 - \frac{\langle \nabla W, \nabla W' \rangle}{\|\nabla W\| \cdot \|\nabla W'\|}$$

Cosine Dissimilarity

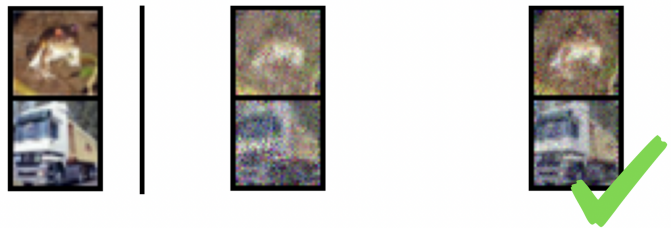


Figure: Different methods for evaluating gradient differnce.

iDLG - Extract Real Labels from Victim

Analytical approach to extract the ground-truth labels.[1]

In cross-entropy loss case:

$$l(x, c) = -\log \frac{e^{y_c}}{\sum_j e^{y_j}}$$

From this, we compute the gradient of the loss for the i^{th} class :

$$g_i = \frac{\partial l(x, c)}{\partial y_i} = \begin{cases} -1 + \frac{e^{y_i}}{\sum_j e^{y_j}} < 0, & \textbf{if i = c}, \\ \frac{e^{y_i}}{\sum_j e^{y_j}} > 0, & \text{otherwise.} \end{cases}$$

Output **y** is not always available.

$\rightarrow$ From the gradient of the last layer :

$$\nabla W_L^i = \frac{\partial l(x, c)}{\partial W_L^i} = \frac{\partial l(x, c)}{\partial y_i} \cdot \frac{\partial y_i}{\partial W_L^i}$$

$$= g_i \cdot a_{L-1}$$

Note : Works for the case where a_{L-1} is always > 0 (ReLU, Sigmoid).

[1]The data can be reconstruct more effectively based on correct labels.

Table of Contents

Model architecture implemented

McMahan CNN

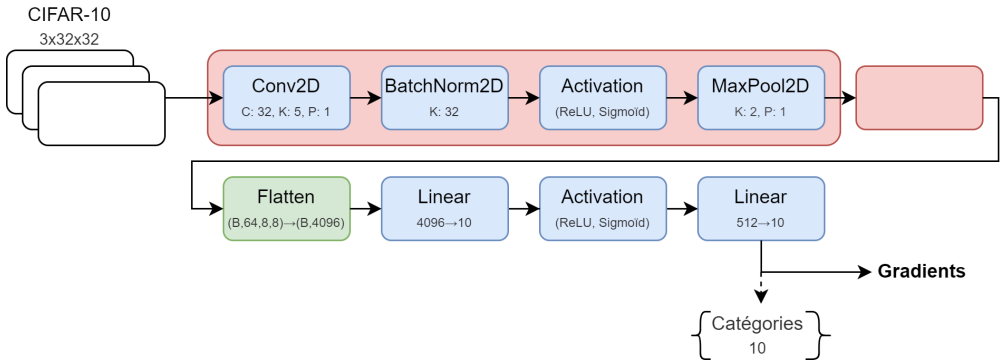


Figure: Architecture of the McMahan CNN model

Model architecture implemented

ResNet

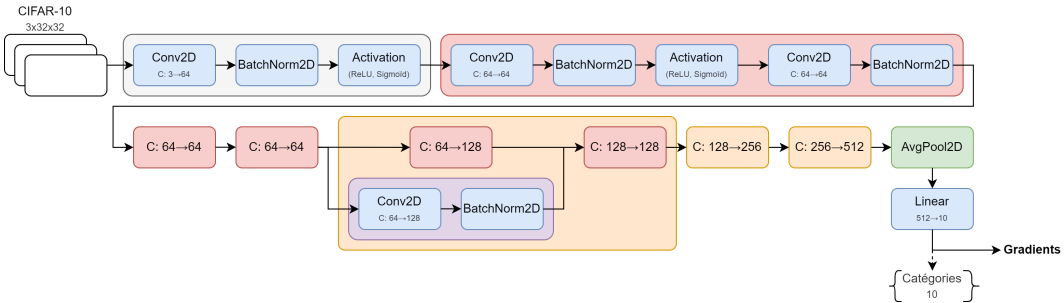


Figure: Architecture of the ResNet-18 model

Table of Contents

Experimental Results

Programming environment

Programming language

Python : Well known language used a lot in AI application.

Framework

PyTorch : Used for AI application.

- Lots of features
- Already optimized
- Can run on GPU

Dataset

CIFAR-10 : Set of 60 000 images (3x32x32) of 10 classes.

Experimental Results

Hydra: Configuration and automatic multi-run

Utility:

- Avoid hard coded variable
- Use configuration files (.yaml)
- Easy to do multi-run

Available configurations:

- Model used (McMahan CNN, ResNET)
- Dataset used (MNIST, CIFAR-10)
- Attacker (Optimizer, Loss, Stop)
- Visualisation (Animation, log scale)
- Data Save (Enable, CSV)

Parameters for our experiments:

- Attacker optimizer : ADAM
- Learning rate : 0.1
- Loss function : Cosine similarity
- Stop criterion : 500 iterations

```yaml
hydra:
  sweeper:
    params:
      +seed: 0, 1 , 2, 3, 4, 5, 6, 7, 8, 9
      model: McMahan_CNN , Custom_ResNet
      batch_size: 1, 2, 4, 8, 16
      model.init_method:
        - xavier_normal_, xavier_uniform_
        - kaiming_normal_, kaiming_uniform_
      model.activation: ReLU, Sigmoid
      client.prune.type: small, null
```

Experimental Results

Others experiments information

- **Total Experiments :**

$Nb_{runs} = 6 \text{ seeds } \times 2 \text{ models } \times 5 \text{ batch size } \times 4 \text{ init methods } \times 2 \text{ activation } \times 2 \text{ prune } = 960$

- **Metric - Image reconstruction quality :**

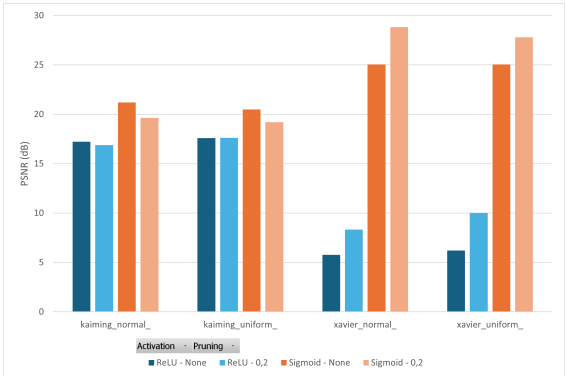
$$PSNR = 10 \cdot \log_{10}\left(\frac{\text{MAX}^2}{\text{MSE}}\right) \quad \text{, where MSE} = \frac{1}{mn}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}\left(I(i,j) - K(i,j)\right)^2$$

- **Weight initialization :**

Kaiming (Normal)	**Kaiming (Uniform)**	**Xavier (Normal)**	**Xavier (Uniform)**
$N(0, std^2)$	$U(-bd, bd)$	$N(0, std^2)$	$U(-a, a)$
$std = \frac{gain}{\sqrt{fan_{mode}}}$	$bd = gain \times \sqrt{\frac{3}{fan_{mode}}}$	$std = gain \times \sqrt{\frac{2}{fan_{in}+fan_{out}}}$	$a = gain \times \sqrt{\frac{6}{fan_{in}+fan_{out}}}$

Experimental Results

Sensitivity to Initialisation Method



Kaiming (Uniform vs. Normal)

- Pruning has low effect
- Best PSNR with Sigmoid

Xavier (Uniform vs. Normal)

- Pruning has a big effect
- Best PSNR with Sigmoid

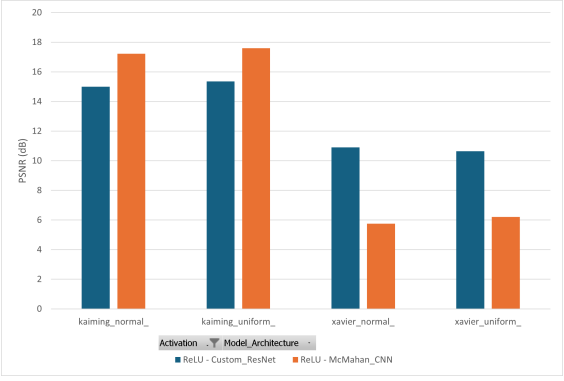
Kaiming vs. Xavier

- Lower PSNR for Xavier with ReLU but better with Sigmoid

Figure: Reconstruction quality of the original image for different **init. methods** (CNN Arch.).

Experimental Results

Sensitivity to Initialisation Method



PSNR depends on the **model architecture**

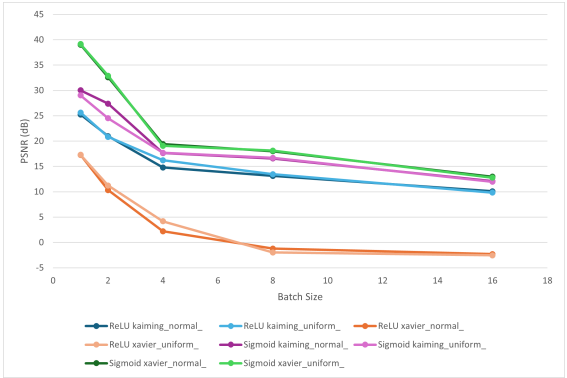
Kaiming vs. Xavier

Lower PSNR for McMahan CNN (ReLU) than ResNet with Xavier weight initialisation.

Figure: Quality of reconstruction of the original image for the **McMahan CNN** and **ResNet** architectures using different **init. methods**.

Experimental Results

Sensitivity to Initialisation Method

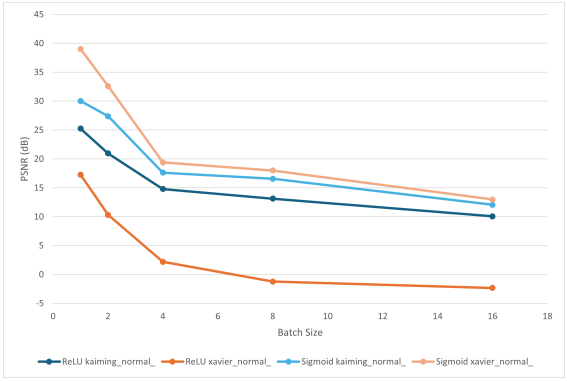


No big variation of **PSNR** between **Uniform** and **Normal** weight initialisation.

Figure: Quality of reconstruction of the original image for different **batch sizes** and **initialization methods** (CNN arch.).

Experimental Results

Sensitivity to Initialisation Method



PSNR **inversely proportional** to number of image per batch.

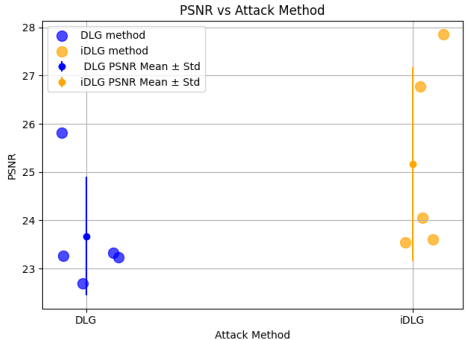
PSNR is the **worst** for ReLu (Xavier)

Large differences in PSNR for **small batch** sizes, but **smaller differences** for **larger batch** sizes.

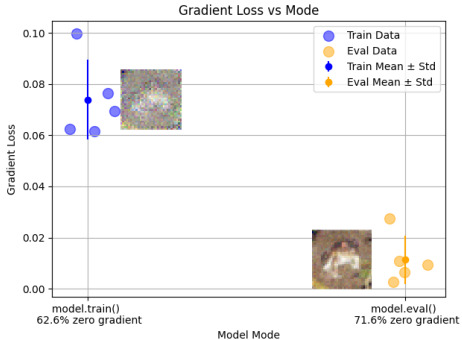
Figure: Quality of reconstruction of the original image for different **batch sizes** and **initialization methods** (CNN arch.) [Simplified].

Experimental Results

Extra Experimental Results



(a) Reconstruction performance by extracted labels.

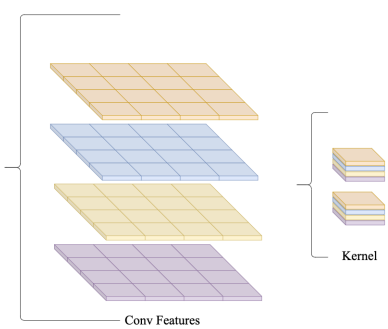


(b) Sensitivity to batch normalization layers.

Figure: On CNN model architecture

Experimental Results

.eval() vs .train() - Sensitivity to Batch Norm Layer



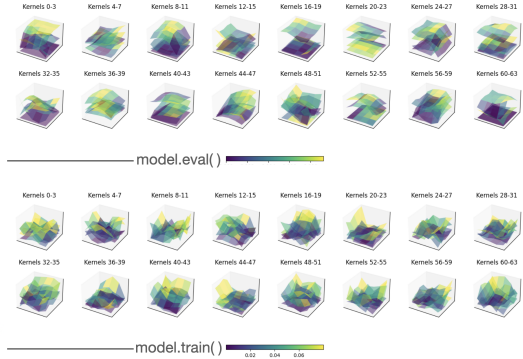
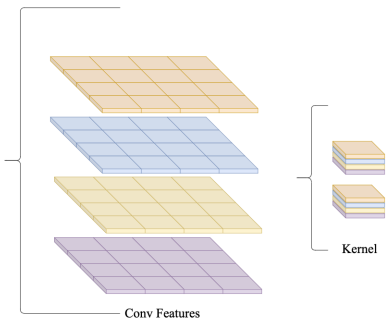


Figure: Visualization of gradient for each kernel.

Experimental Results

.eval() vs .train() - Sensitivity to Batch Norm Layer



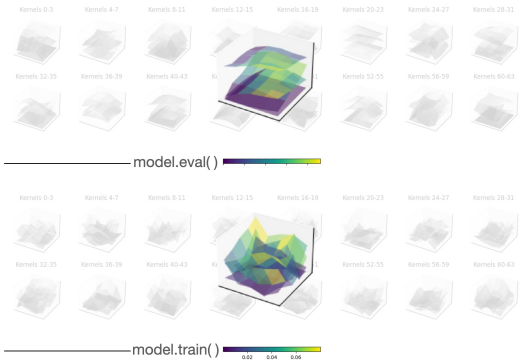


Figure: Visualization of gradient for each kernel.

Defense Strategies

Encryption, Noise, Sparsification

What are the other means of defense ?

- Adding Noise (Gaussian or Laplacian) to gradient before sharing
- Using Higher image resolution
- Using sparcification (over 20 %)
- Using Cryptology (the most secure but cannot be implemented on all model)

Table of Contents

5 Conclusion

Conclusion

Contribution

Model **McMahan CNN** has lower PSNR than **ResNet**

Worst PSNR to best PSNR:
Xavier - ReLU $<$ Kaiming - ReLU $<$ Kaiming Sigmoid $<$ Xavier - Sigmoid

ReLU has the best protection (worst PSNR) because of the $\max()$ function.
There is a **loss of information**.

Table of Contents

Video Show

Video: Reconstruction of Training Data

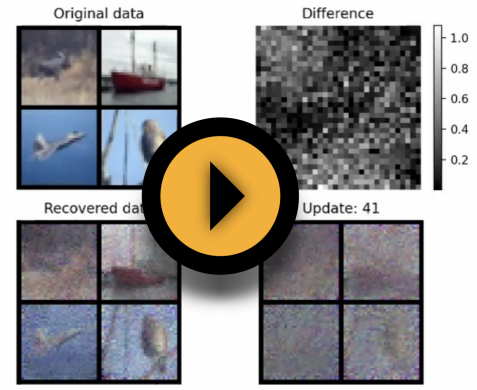


Figure: Training data are reconstructed from the noise images

Reference

1. H. Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, Blaise Agüera y Arcas (2016). Communication-Efficient Learning of Deep Networks from Decentralized Data. arXiv:1602.05629
2. Ligeng Zhu, Zhijian Liu, Song Han (2019). Deep Leakage from Gradients. arXiv:1906.08935
3. Bo Zhao, Konda Reddy Mopuri, Hakan Bilen (2020). iDLG: Improved Deep Leakage from Gradients. arXiv:2001.02610.

Attacking gradients: reconstructing training data

Thank you for listening!
Any questions?