Robust Federated Learning

Bill Tao
Federated learning

- Personal data is stored on local devices (A)
- Each device train the model locally and return a version of model parameters (B)
- The parameters are aggregated at the central server (C)
Towards Federated Learning With Byzantine-Robust Client Weighting

A. Portnoy et.al.
Byzantine clients

- Byzantine fault: the server doesn't know if a client is malfunctioning
- The server relies on the clients to report the number of samples and training result
- A client can provide fake number of data samples AND adverse content in the samples

N=3
N=5
N=4
N=2,147,483,647
Robustness through truncation

- Core idea: **Don’t let 1% clients provide 99% of the data!**
  - “Nobody can have more than U samples!”
- How do we determine U?
  - We don’t want a few clients to take up the majority of data
  - Maximum weight proportion: proportion of the most weighted clients
  - Goal: \( mwp(\text{truncate}(N,U),p) < \alpha^* \) after truncation
Solve for optimal cut-off

- Express mwp as
  \[ \frac{\sum_{(1-\alpha)K<i\leq u} n_i + \{n_i : i > \max(u, (1-\alpha)K)\}}{\sum_{i\leq u} n_i + \{n_i : i > u\}} U \]

- Solve U:
  \[ U^* \left\langle \frac{a - c\alpha^*}{d\alpha^* - b} \right\rangle \]

- Trade-off: the larger \( \alpha \) is, the lower U can be
In practice..

- Total number of clients is large
- Solve $U$ using a sample from $N$ clients
- How confident are we on the solution?

**Theorem 3.1.** Given parameter $\delta > 0$ and $\varepsilon_1 = \sqrt{\frac{\ln (3/\delta)}{2k}}$, $\varepsilon_2 = U\sqrt{\frac{\ln \ln (3/\delta)}{2(k(\alpha-\varepsilon_1)+1)}}$, $\varepsilon_3 = U\sqrt{\frac{\ln \ln (3/\delta)}{2k}}$, we have that $\text{mwp} (\text{trunc}(N, U), \alpha) \leq \alpha^*$ is true with $1 - \delta$ confidence if the following holds:

$$\alpha \left( \frac{\sum_{i \leftarrow \text{trunc}(N, U)} X_{(i)} + \varepsilon_2}{\frac{1}{k} \sum_{i \in [k]} X_i - \varepsilon_3} \right) \leq \alpha^*$$ (6)
Influence on optimization goal

- The error for loss function estimation is bounded

\[
\left\| \frac{1}{\tilde{n}} \sum_{i \in [K]} \hat{n}_i F_i(w) - \frac{1}{\tilde{n}} \sum_{i \in [K]} \tilde{n}_i F_i(w) \right\| \leq \\
\left\| \sum_{i: \hat{n}_i > U} \left( \frac{\hat{n}_i}{\tilde{n}} - \frac{1}{K} \right) F_i(w) + \left( \frac{1}{\hat{n}} - \frac{1}{\tilde{n}} \right) \sum_{i: \hat{n}_i \leq U} \mathcal{L}(Z_i) \right\|
\]

True loss

Unbalancedness

Estimated loss

Truncation error
Evaluation

Testbed
- Dataset: Shakespeare, next-character prediction
- Model: LSTM

Setup
- Server: trust all clients (passthrough), truncate the numbers or distrust all clients (treat them as equal weight)
- Attack: Model negation attack (pushing model parameter to 0) and Label shifting attack (shifting the predicted label)
Evaluation

Passthrough doesn’t work

Better considering client weight than not
Comments

- Intuitive solution
- Needs more analysis on influence on convergence
  - Theorem 3.2 (loss function error bound) is not enough because it does not tell about the difference between ground truth and the proposed method
- It’s not persuasive that we need to solve U using partial information of N
DBA: Distributed Backdoor Attacks Against Federated Learning

C. Xie et.al.
Backdoor attack

- Corrupt the training dataset
  - Adding *trigger* to the training input images
  - Changing *label* for those images to a desired one

- Result:
  - The model behave normally otherwise
  - When trigger is present (regardless of the true image), the model gives expected prediction
Distributed backdoor attack

- Decompose the *global* trigger into *local* triggers
- Each attacker only injects one local trigger
Mathematical formulation

Original backdoor attacker:

\[ w_i^* = \arg \max_{w_i} \left( \sum_{j \in S^i_{poi}} P[G^{t+1}(R(x_j^i, \phi)) = \tau] + \sum_{j \in S^i_{cln}} P[G^{t+1}(x_j^i) = y_j^i] \right) \]

Polluted data predicted wrong

Normal data predicted right

Transformation to add trigger

Trigger

Target label

Distributed backdoor attacker:

\[ w_i^* = \arg \max_{w_i} \left( \sum_{j \in S^i_{poi}} P[G^{t+1}(R(x_j^i, \phi_i^*)) = \tau; \gamma; I] + \sum_{j \in S^i_{cln}} P[G^{t+1}(x_j^i) = y_j^i] \right) \]

Local triggers
Evaluation: setup

- 4 datasets
  - LOAN
  - MNIST
  - CIFAR-10
  - Tiny Imagenet
- Comparing DBA vs centralized
  - Single shot vs multi shot (attackers inject triggers across several epochs)
- Defense testbeds:
  - DFA: suppress outliers
  - FoolGold: suppress clients repeatedly submitting same gradients
Evaluation: no defence

DBA More persistent

Global trigger always better than local
Evaluation: against DFA/FoolsGold

DBA performs better
Ablation study

Individual local trigger has low excitation

Global trigger drags attention
Case study: effects of trigger features

Doesn’t work when trigger too small

Break the main model when trigger too large

Doesn’t work when trigger overlap with the center area
Case study: effects of trigger features (cont’d)

Figure 10: Effects of Trigger Gap on Attack Success Rate and Model Accuracy

Figure 11: Effects of Local Trigger Size on Attack Success Rate and Model Accuracy
Case study: effects of trigger features (cont’d)

Optimal position round exists

Doesn’t work if too little data poisoned

Too much poison blows up main model

Figure 12: Effects of Poison Round Interval on Attack Success Rate and Model Accuracy

Figure 13: Effects of Poison Ratio on Attack Success Rate and Model Accuracy
Comments

- Novel idea to address an important issue
- Extensive ablation study & case study
  - Clear explanation of why trigger features influence success rate
- Can have more evaluation:
  - Different number of adversarial parties, etc.
Questions?