3 Capstone Project: Adversarial Federated Learning

3.1 Description

Federated learning is introduced to enhance users' privacy by performing training using local data but only sharing model parameter update with the server. Unlike traditional machine learning algorithm that requires centralizing data, federated learning achieves training by aggregating updates from users without peeking at users' data. However, this gives rise to a new threat called model poisoning, where a malicious client is aimed to cause the global model to misclassify a set of given inputs with high confidence by sending a fake update. In this project, assuming the server is using a deep neural network on an image classification task, you will explore the vulnerability of the update aggregation step of federated learning and devise strategies to prevent or mitigate those attacks.

As Malicious Clients, you need to make sure that:

- Data will be distributed randomly and uniformly
- Global model at the server needs to convergence
- No or Little damage on the performance of the global model
- High attacking successful rate on the testing data that is not accessible by any clients
- Constant attacking successful rate after several epochs of training

For Protector, here is a very general tip for detecting malicious clients:

- Malicious clients often make significant changes or boosting to their update in order to impact the global model. You goal is to find anomalies among benign updates.

3.2 Datasets and Models

You can use MNIST or CIFAR10 for your experiments. (*http://yann.lecun.com/exdb/mnist/* and *https://www.cs.toronto.edu/ kriz/cifar.html*) Regarding model selection, you have a wide choice of Lenet, Densenet, EfficientNet, MobileNet, etc. Refer to this link for state-of-the-art networks on these datasets.
(*https://paperswithcode.com/sota/image-classification-on-mnist*)

3.3 Submission Requirement

- Jupyter notebooks (ipynb) or python source code for training, inference, and evaluation of your model.

  - You may choose Tensorflow or PyTorch for implementation
  - implementation must be Python-based
- Instructions to set up and run the source code (Docker file, shell script, pip commands, etc.)
- PowerPoint slides and the corresponding PDF files illustrating the graph embedding method
- Any datasets you used or links to download the datasets
- Report with the results supporting your comparative analysis: include the following information and metrics at a minimum:
  - Description of the datasets
  - Description of the specific problem of your choice.
  - Description of any assumption you make. (e.g. What data does the clients have access to? What is the number/percentage of malicious clients in total?)
  - Description of the specific algorithm
  - Architectures Hyper-parameters: table
  - Hyper-parameter Tuning: Choices, rationale, observed impact on the model
  - Plots of loss, global accuracy, attacking successful rate along with training
  - Detailed comparison between non-attacked model and attacked model
  - Reasoning about why your attacking/detection works

3.4 Useful Resources:

- PySyft is a Python library for secure and private Deep Learning. *https://github.com/OpenMined/PySyft*
- Analyzing Federated Learning through an Adversarial Lens. *https://www.princeton.edu/ pmittal/publications/bhagoji-icml19.pdf*
- Can You Really Backdoor Federated Learning? *https://arxiv.org/pdf/1911.07963.pdf*
- Learning to Detect Malicious Clients for Robust Federated Learning *https://arxiv.org/pdf/2002.00211.pdf*
- Backdoor Embedding in Convolutional Neural Network Models via Invisible Perturbation *https://arxiv.org/pdf/1808.10307. pdf*