pFedDef: Defending Grey-Box Attacks for Personalized Federated Learning

This repository is the official implementation of the NeurIPS 2022 submission of the same name.

Summary:

Personalized federated learning allows for clients in a distributed system to train a neural network tailored to their unique local data while leveraging information at other clients. However, clients' models are vulnerable to attacks during both the training and testing phases. In this paper we address the issue of adversarial clients crafting evasion attacks at test time to deceive other clients. For example, adversaries may aim to deceive spam filters and recommendation systems trained with personalized federated learning for monetary gain. The adversarial clients have varying degrees of personalization based on the method of distributed learning, leading to a "grey-box" situation. We are the first to characterize the transferability of such internal evasion attacks for different learning methods and analyze the trade-off between model accuracy and robustness depending on the degree of personalization and similarities in client data. We introduce a defense mechanism, pFedDef, that performs personalized federated adversarial training while respecting resource limitations at clients that inhibit adversarial training. Overall, pFedDef increases relative grey-box adversarial robustness by 62% compared to federated adversarial training and performs well even under limited system resources.

The code in this repository has been written to implement the pFedDef algorithm and perform system analysis regarding pFedDef. In the context of this repository, pFedDef is equivalent to the FedEM_adv setting.

Requirements

To install requirements:

```
pip install -r requirements.txt
```

Data Sets

The CIFAR-10 and CIFAR-100 data sets are readily available to set up with the existing infrastructure of FedEM. The CelebA data set can be found and downloaded at LEAF, while the MovieLens data set is found at MovieLens.

Original Repository

The code from this repository has been heavily adapted from: Federated Multi-Task Learning under a Mixture of Distributions. The code is found at https://github.com/omarfoq/FedEM.

The following components of their work has been used as a basis for our work.

- data folder with the data set downloading and data splitting
- learner folder with learner and learners_ensemble classes
- 'client.py', 'aggregator.py' classes

Our Contribution

We have added the following unique files for experiments:

- Transfer_attacks folder and contents
  - Transfer attacks between one client to another in a federated learning setting
  - Boundary transferer used to measure inter-boundary distance as from: The Space of Transferable Adversarial Examples
- 'solve_proportions()' function from 'Transfer_attacks/TA_utils.py' solves the robustness propagation problem given limited resources.

The following aspects of the FedEM code has been altered:

- 'client.py' now has adversarial training mechanisms, more detailed implementations of local tuning, and label swapping sybil attack mechanism
- 'Aggregator.py' has implementation of KRUM from Machine Learning with Adversaries: Byzantine Tolerant Gradient Descent (work in progress)

Training

The model weights can be trained given varying methods of adversarial training and aggregation method by running the .py files in the 'run_experiments_collection' folder. Move the .py file to the root directory to run. Inside the file, there is a section of inputs.

The following inputs are of importance:

- G: Adversarial dataset proportion globally
- Q: Number of rounds between adv data generation
- K: Number of steps used to perturbed dataset when generating adversarial examples
- eps: Magnitude of projection during projected gradient descent
- args_.experiment: type of dataset to run (cifar10, cifar100, celeba)
- args_.method: training method to use ('FedAvg', 'FedAvg_adv', 'FedEM', 'FedEM_adv', 'local', 'local_adv')
- args_.n_rounds: Number of rounds to run training for
- args_.save_path: Location to save weights

The scripts have been written in such a way that an individual script can be run for consecutive training of related neural networks. All experiments require NVIDIA GPU and CUDA library, and has been run on AWS EC2 g4dn.xlarge instance.

Evaluation

The evaluation of saved neural networks are performed in jupyter notebook instances found in the Evaluation folder. Individual notebooks load relevant weights and perform adversarial attacks on the models. Note that the jupyter notebook environment and package dependency is equivalent to the .py files used to run the experiments.

The following evaluation tools are included:

- loading a pre-trained group of federated learning models for different learning types, and performing transfer attack between clients and recording statistic
- performing ensemble attack, where multiple clients jointly perform attacks by sharing gradient information as seen in Ensemble adversarial black-box attacks against deep learning systems
- Performing inter-boundary distance measurements between different models.