LDP-FL: Practical Private Aggregation in Federated Learning with Local Differential Privacy 2021

IJCAI Lichao Sun, Jianwei Qian, Xun Chen

Contributions

1. First, existing approaches have assumed a fixed range of weights for simplicity.
2. Second, when clients uploads the local model to the server, the server can explore the private connections of model weights, which causes the privacy budget explosion due to the high dimensionality of deep learning models.

- Author propose a new data perturbation with adaptive range by considering that model weights at different deep neural network (DNN) layer could vary significantly.

- Author propose a parameter shuffling mechanism to each clients' weights to mitigate the privacy degradation caused by the high data dimensionality of deep learning models and many query iterations.

Method

- Data Perturbation with Adaptive Range

(1) All clients and the server agree to the same weight range, represented by (C_0, R_0) based on prior knowledge at the beginning when initializing weights.
(2) In local update, based on the weight range (C_l, R_l) of each layer where $C_l \in C$and $e_l \in R$, each local client can perturb its weights by the proposed LDP data perturbation given by
(3) In cloud update, the cloud calculates and updates C and R from the local weights update received from clients. The updated C and R will be distributed to the clients when the cloud distributes the updated weights.

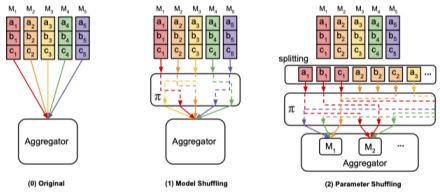
Algorithm 2: DataPertubation

Input: Original local weights W^s_{l+1}, range represented by C_l and R_l, privacy budget ϵ
Output: Perturbed weights W^s_{l+1}

for *each* $w \in W^s_{l+1}$ *and corresponding* $c \in C_l$ *and* $r \in R_l$ **do**
  Sample a Bernoulli variable u such that
  $\Pr[u = 1] = \frac{(w-c)(e^\epsilon-1)+r(e^\epsilon+1)}{2r(e^\epsilon+1)}$;
  if *u==1* **then**
    $w^* = c + r \cdot \frac{e^\epsilon+1}{e^\epsilon-1}$;
  else
    $w^* = c - r \cdot \frac{e^\epsilon+1}{e^\epsilon-1}$;
  $w \leftarrow w^*$; % update W^s_{l+1};
return.

- Parameter Shuffling,To solve the curse of the high dimensionality of DNN models, parameter shuffling is executed in two steps.

(1) In the first step, each client splits the weights of their local model, but labels each weight with an id to indicate its location of the weight in the network structure.
(2) In the second step, each client samples a small random latency t from a uniform distribution $U(0, T)$, where $T > 0$, for each weight and waits for t before sending the weight to the cloud.



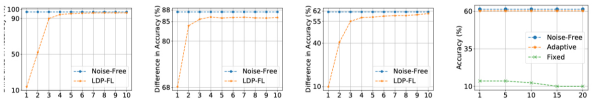
Algorithm 3: ParameterShuffling

Input: Perturbed weights W^s_{l+1} after Algorithm 2

label the position id of each element of W;
for *each element* $w^s \in W$ **do**
    label the element position with a unique id
    $t^s_{id} \leftarrow U(0, T)$ % Randomly sample a small latency between 0 and T;
$SendToCloud(id, w_{id})$ at time t^s_{id};
return.

Experimental

- MNIST/FMNIST/GPU NVIDIA Tesla V100



(a) Effect of ϵ on MNIST (b) Effect of ϵ on FMNIST (c) Effect of ϵ on CIFAR-10 (d) Effect of r on CIFAR-10