Possible Algorithms for Federated Learning

Jinming Xu

January 13, 2020

1 Problem

Given n observations $(x_i, y_i) \in \mathcal{X} \times \mathcal{Y},\ i = 1, 2, ..., n$, we assume the data is partitioned into K datasets to be maintained at each device (client), with $\mathcal{P}_k$ the set of indexes of data points on client $k \in \{1, 2, ..., K\}$ and $n_k = |\mathcal{P}_k|$ the number of data points kept at client k. Note that we have $\sum_{i=1}^{K} n_i = n$.

We attempt to solve the following probelm [1]

$$\min_w f(w) := \sum_{i=1}^{n} f_i(w) \tag{1}$$

with $f_i(w) \triangleq \sum_{j \in \mathcal{P}_i} l(w; x_j, z_j)$. The next section provides some possible parallel algorithms adapted from distributed conterparts for solving the above problem in a way that the dataset is not disclosed during the whole training process.

2 Possible Algorithms

2.1 Primal-only Methods

The variation among the dataset can be regarded as another level of "stochasity", for which variance-reduced methods can be applied. The following algorithm is adapted from Push-Pull [2], which shares some similarity with SAGA [3].

Algorithm 1 Parallel "SAGA"

Initialization $y_i^0 = \nabla f_i(w^0), i = 1, 2, ..., K;\ y^0 = \sum_{i=1}^K y_i^0$

for $k = 0, 1, ..., T$ **do**

SeverUpdate: ▷ run on sever with variables w, y

1: $S_k \leftarrow$ {random (or selected) set of clients}

2: **for** each client $j \in S_k$ **in parallel do**

$$\Delta g_j^k = \textbf{ClientUpdate}(j, w^k)$$

3: **end for**

4: Update w, y using $\{\Delta g_j^k\}$ as

$$y^{k+1} = y^k + \sum_{j \in S_k} \Delta g_j^k$$
$$w^{k+1} = w^k - \eta y^{k+1}$$

(note that y^{k+1} can be regarded as an estimate of the global gradient $\sum_{i=1}^K \nabla f_i(w^k)$)

ClientUpdate(j, w)**:** ▷ run on client j with variable y_j

if $j \in S_k$ **then**

return the incremental gradient change $\nabla f_j(w) - y_j^k$ to the server

$y_j^{k+1} = \nabla f_j(w)$; store y_j^{k+1}

else

$y_j^{k+1} = y_j^k$

end if

end for

Return: w^T

The above algorithm needs the full gradient of each local dataset, which is usually computationally prohibitive. Thus, the following provides the stochastic/batch version. Note that the server still maintain w, y while each client k is now taking charge of a set of variables $\{y_i\}, i \in \mathcal{P}_k$ corresponding to each data point kept at client k.

Algorithm 2 Stochastic/Batch Parallel "SAGA"

Initialization $y_i^0 = \nabla l(w^0; x_i, z_i), i = 1, 2, ..., n;\ y^0 = \sum_{i=1}^{n} y_i^0$

for $k = 0, 1, ..., T$ **do**

SeverUpdate: ▷ run on sever with variables w, y

1: $S_k \leftarrow$ {random (or selected) set of clients}

2: **for** each client $j \in S_k$ **in parallel do**

$$\Delta g_j^k = \textbf{ClientUpdate}(j, w^k)$$

3: **end for**

4: Update w, y using $\{\Delta g_j^k\}$ as

$$y^{k+1} = y^k + \sum_{j \in S_k} \Delta g_j^k$$
$$w^{k+1} = w^k - \eta y^{k+1}$$

ClientUpdate(j, w)**:** ▷ run on client j with a table of variables $\{y_i\}, i \in \mathcal{P}_j$

if $j \in S_k$ **then**

select a batch $\mathcal{B} \subseteq \mathcal{P}_j$ of size B uniformly at random

return the incremental gradient change $\sum_{j \in \mathcal{B}} (\nabla l(w; x_j, z_j) - y_j^k)$ to the server

$y_j^{k+1} = \nabla f_j(w)$ and store y_j^{k+1} for all $j \in \mathcal{B}$

else

$y_j^{k+1} = y_j^k$

end if

end for

Return: w^T

2.2 Primal-Dual Methods

The following is another possible method to solve the above problem from the ADMM framework [4], which is obtained by setting $W = \frac{\mathbf{1}^T\mathbf{1}}{K}$ of ID-FBBS schemes in [5].

Algorithm 3 Parallel "ADMM"

Initialization: $y_i^0 = 0,\ i = 1, 2, ...K$

for $k = 0, 1, ..., T$ **do**

SeverUpdate($\{w_i^k\}, K$):

1: gather $\{w_i^k\}$ from all clients and calculate the average

$$w_{av}^k = \frac{1}{K}\sum_{i=1}^{K} w_i^k.$$

ClientUpdate(w_{av}^k, η):

1: **for** each client $i = \{1, 2, ..., K\}$ **do**

$$w_i^{k+1} = w_{av}^k - \eta \nabla f_i(w_i^k) - y_i^k$$
$$y_i^{k+1} = y_i^k + w_i^{k+1} - w_{av}^k$$

2: **end for**

end for

Return: w_{av}^T

As before, the above algorithm needs to evaluate the full gradient of local datasets at each iteration. The following provides the stochastic/batch version.

Algorithm 4 Stochastic/Batch Parallel "ADMM"

Initialization: $\sum_{i=1}^K y_i^0 = 0$

SeverUpdate($\{w_i^k\}, K$):

1: gather $\{w_i^k\}$ from all clients and calculate the average

$$w_{av}^k = \frac{1}{K}\sum_{i=1}^{K} w_i^k.$$

ClientUpdate(w_{av}^k, η):

1: **for** $i = \{1, 2, ..., K\}$ **do**

2: select a batch $\mathcal{B} \subseteq \mathcal{P}_j$ of size B uniformly at random

$$w_i^{k+1} = w_{av}^k - \eta \sum_{j \in \mathcal{B}} \nabla l(w_{i\ \text{av}}^k; x_j, z_j) - y_i^k$$
$$y_i^{k+1} = y_i^k + w_i^{k+1} - w_{av}^{k+1}$$

3: **end for**

Remark 1. *The above term highlighted in red color may be replaced with some estimate* *with reduced variance, similar with the idea of SAGA.*

References

[1] H. B. McMahan, E. Moore, D. Ramage, S. Hampson *et al.*, "Communication-efficient learning of deep networks from decentralized data," *arXiv preprint arXiv:1602.05629*, 2016.

[2] S. Pu, W. Shi, J. Xu, and A. Nedić, "A push-pull gradient method for distributed optimization in networks," in *2018 IEEE Conference on Decision and Control (CDC)*. IEEE, 2018, pp. 3385–3390.

[3] A. Defazio, F. Bach, and S. Lacoste-Julien, "Saga: A fast incremental gradient method with support for non-strongly convex composite objectives," in *Advances in Neural Information Processing Systems*, 2014, pp. 1646–1654.

[4] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundations and Trends in Machine Learning*, vol. 3, no. 1, pp. 1–122, Jan 2011.

[5] J. Xu, S. Zhu, Y. C. Soh, and L. Xie, "A bregman splitting scheme for distributed optimization over networks," *IEEE Transactions on Automatic Control*, vol. 63, no. 11, pp. 3809–3824, 2018.