Large-Scale Distributed Private Learning with DPSGD

Tianhao Wang, Litao Yan, Ruoxi Yang, Silin Zou

Overview

- Type of application
  - Data Preprocessing
  - DPSGD Algorithm
- Levels of parallelism
  - Big data
  - Distributed parallelization
- Programming model
  - Spark
  - Distributed package of PyTorch
    - with MPI backend
- Parallel algorithm
  - Synchronous Fashion
  - Asynchronous Fashion
  - Ring All-Reduce
- Infrastructure
  - AWS 4 g3.4xlarge

MPI

Algorithm 1 Differentially private SGD (Outline)

Input: Examples $\{x_1, \ldots, x_N\}$, loss function $\mathcal{L}(\theta) = \frac{1}{N}\sum_i \mathcal{L}(\theta, x_i)$. Parameters: learning rate η_t, noise scale σ, group size L, gradient norm bound C.

Initialize θ_0 randomly

for $t \in [T]$ **do**

Take a random sample L_t with sampling probability L/N

Compute gradient

For each $i \in L_t$, compute $\mathbf{g}_t(x_i) \leftarrow \nabla_{\theta_t}\mathcal{L}(\theta_t, x_i)$

Clip gradient

$\bar{\mathbf{g}}_t(x_i) \leftarrow \mathbf{g}_t(x_i) / \max\left(1, \frac{\|\mathbf{g}_t(x_i)\|_2}{C}\right)$

Add noise

$\tilde{\mathbf{g}}_t \leftarrow \frac{1}{L}\left(\sum_i \bar{\mathbf{g}}_t(x_i) + \mathcal{N}(0, \sigma^2 C^2 \mathbf{I})\right)$

Descent

$\theta_{t+1} \leftarrow \theta_t - \eta_t \tilde{\mathbf{g}}_t$

Output θ_T and compute the overall privacy cost (ε, δ) using a privacy accounting method.

Spark

	sex	married	black	asian	collegedegree	employed	militaryservice	uscitizen	disability	englishability	blackfemale
0	1	1	0	0	0	1	0	1	0	1	0
1	1	1	0	0	0	0	0	1	0	1	0
2	0	1	0	0	0	0	1	1	0	1	0
3	0	0	0	0	0	0	0	1	1	1	0
4	0	1	0	0	0	0	0	1	0	1	0

Sequential Code Analysis

```python
if __name__=='__main__':
    model = Network()
    model.to(device)

    l2_norm_clip = 3
    noise_multiplier = 0.9
    batch_size = 256
    minibatch_size = 3

    optimizer = DPSGD(
        params = model.parameters(),
        l2_norm_clip = l2_norm_clip,
        noise_multiplier = noise_multiplier,
        batch_size = batch_size,
        minibatch_size = minibatch_size,
        lr = 0.01,
    )

    DPtrain(model, device, DIR+'CaPUMS5full.csv', optimizer, epoch_nb=50,
        target_acc=0.7)
```

Acc: 0.67

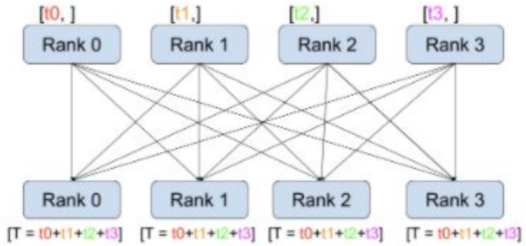
Real: 16m 32s User: 15m 16s Sys: 33s

Profiling:

```
123912621 function calls (121671800 primitive calls) in 991.690 seconds
```

```python
for epoch in range(epoch_nb):
  batch_loss = 0
  batch_acc = 0

  for i in range(batches_per_epoch):
    idx = np.random.randint(0, x_train.shape[0], batch_size)
    x_train_batch, y_train_batch = x_train[idx], y_train[idx]

    optimizer.zero_grad()

    for _ in range(minibatches_per_batch):
      idx = np.random.randint(0, batch_size, minibatch_size)
      x_train_mb, y_train_mb = x_train_batch[idx].to(device), y_train_batch[idx].to(device)

      optimizer.zero_minibatch_grad()
      pred_mb = model(x_train_mb)
      loss = lossFnc(pred_mb, y_train_mb.unsqueeze(1))
      acc = binary_acc(pred_mb, y_train_mb.unsqueeze(1))
      loss.backward()
      optimizer.minibatch_step()
```

```
1109350     9.492     0.000   136.543     0.000 main.py:25(forward)
1109350    25.104     0.000    83.242     0.000 main.py:32(binary_acc)
```



All-Reduce

parallelization overheads

- Spark: Spark RDD persistence (caching)
  - Use cached RDDs for reuse, avoid reloading dataset everytime it is used
  - Decreases the computation time by almost 100x, comparing to other distributed computation frameworks (MapReduce)
- Reduce overhead on torch.distributed
  - Load balancer: solve the imbalanced issue while using mixed GPUs
  - Data partition: reduce communication betwee nodes
  - Communication: Point to Point & Collective Communication
    - centralized vs decentralized (ring all-reduce)

Runtime Analysis

In regular algorithm, we measure the numerical complexity of a problem as a function of input size, but for a learning algorithm, it does not make sense to define the input size to be the size of training set.

Theorem 1. *Assume there are m workers, each with a set of data $S_i, i \in [m]$. Suppose the sensitivity bound of gradient specified in DPSGD is C. Let $B > 0$. Suppose the target loss function f is convex, let $w^* = arg\min_{||w|| \leq B} f(w)$. Assume DPSGD algorithm is run for T iterations with learning rate $\eta = \sqrt{\frac{B^2}{C^2 T}}$, then*

$$E[f(\bar{w})] - f(w^*) \leq \frac{BC}{\sqrt{mT}}$$

Hence, suppose we are aiming for ϵ accuracy for the learned classifier, then it suffices to run DPSGD for a number of iterations that satisfies

$$T \geq \frac{B^2 C^2}{m\epsilon^2}$$

Shalev-Shwartz, Shai, and Shai Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.
Zinkevich, Martin, et al. "Parallelized stochastic gradient descent." *Advances in neural information processing systems*. 2010.