Large-Scale Distributed Private Learning with DPSGD

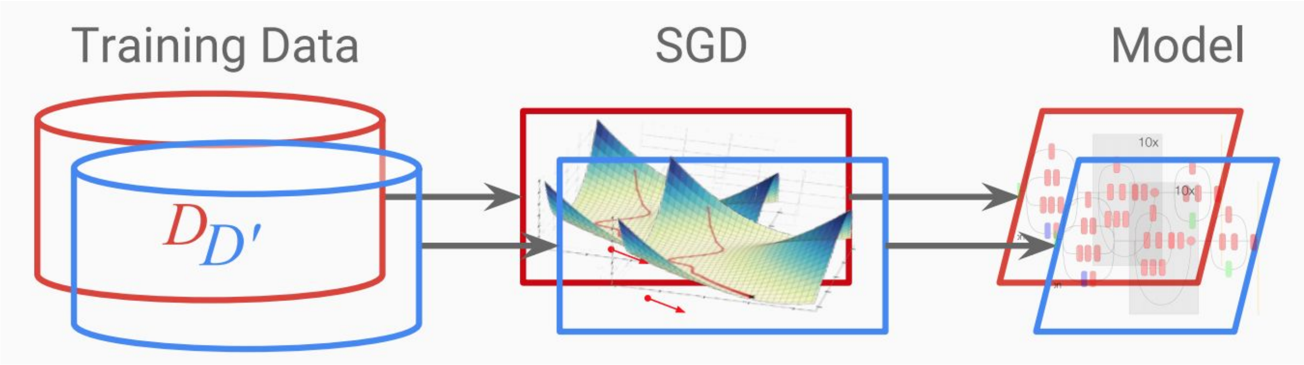
Tianhao Wang, Litao Yan, Ruoxi Yang, Silin Zou

Project Website: https://yanlitao.github.io/fastDP/

Project Repo: https://github.com/TIANHAO-WANG/fastDP

Overview

Private Data

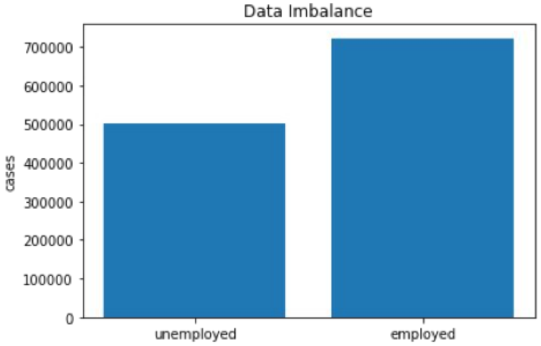


Spark

Distributed Training

Data Processing: technique

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



Spark vs. MapReduce

- In-Memory vs. I/O on Disk
- Ease of Use
- Suitable for our dataset

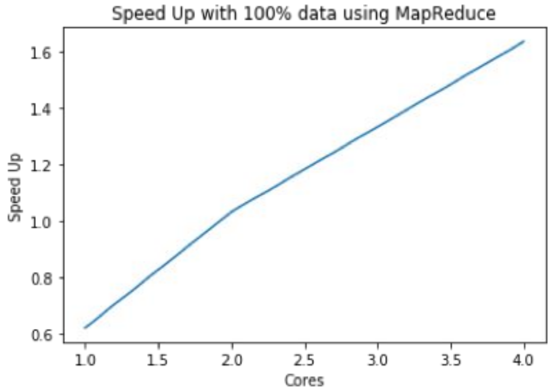
Data Processing: Result

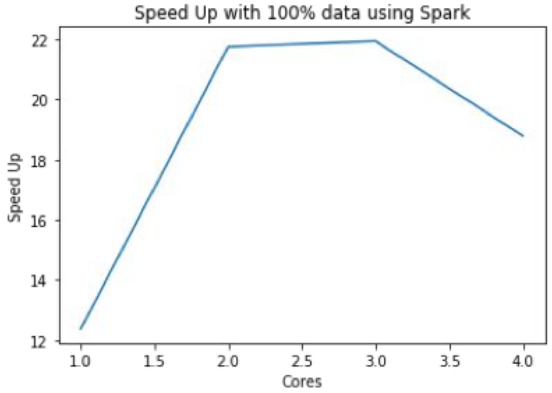
MapReduce

Percentage of Full Dataset	Sequential Time	with 1 core	with 2 cores	with 4 cores
1%	0.927s	110s	66s	44s
25%	17.579s	114s	68s	44s
100%	78.52s	126s	76s	48s

Spark

Percentage of Full Dataset	Sequential Time	with 1 core	with 2 cores	with 3 cores	with 4 cores
100%	78.52s	6.34s	3.61s	3.58s	4.18s





Distributed DPSGD Training: Design

Choices:

- Model Parallel vs **Data Parallel**
- Parameter Server vs **AllReduce**
- CPU vs **GPU-accelerated**

Avoid Data Communication

Avoid model synchronization

Algorithm 1 Distributed Parallelization of Differentially private SGD

Input: Training Data $\{x_1, \ldots, x_N\}$, loss function $\mathcal{L}(\theta) = \frac{1}{N}\sum_i \mathcal{L}(\theta, x_i)$. Training hyperparameters: learning rate η_t, noise scale σ, gradient norm bound C, total number of iterations T. Parallelization parameters: individual batch size B, number of workers R.

Initialize θ_0 randomly with the same seed for all R workers.

Partition training data into R pieces and distribute to each worker.

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

Take a random batch $L_{t,r}$ of size B on the local data for each worker r

Compute local gradient

For each $i \in L_{t,r}$, compute $\mathbf{g}_{t,r}(x_i) \leftarrow \nabla_{\theta_t}\mathcal{L}(\theta_t, x_i)$

Clip local gradient

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

Noise Addition

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

Gradient AllReduce

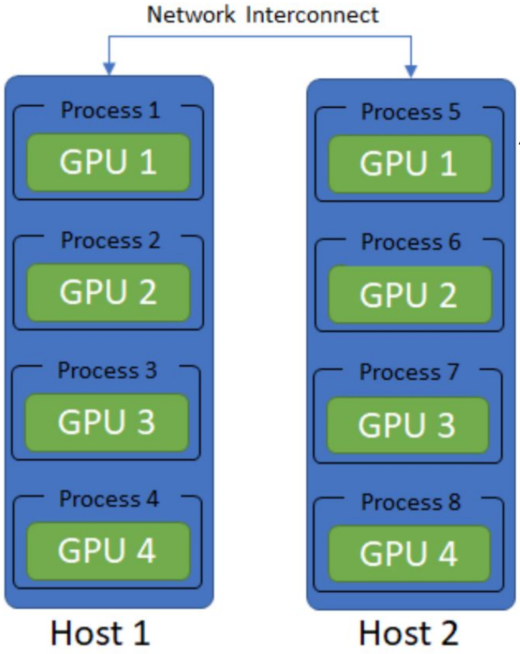
sum all $\tilde{\mathbf{g}}_{t,r}$ across all of the workers into $\tilde{\mathbf{g}}_t$.

Update $\theta_{t+1} \leftarrow \theta_t - \eta_t \tilde{\mathbf{g}}_t / R$ on all workers.

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

AllReduce = Reduction + Broadcast

Distributed DPSGD Training: Design



Cl

Avoid Data Communication

el

ice

Avoid model synchronization

Algorithm 1 Distributed Parallelization of Differentially private SGD

Input: Training Data $\{x_1, \dots, x_N\}$, loss function $\mathcal{L}(\theta) = \frac{1}{N}\sum_i \mathcal{L}(\theta, x_i)$. Training hyperparameters: learning rate η_t, noise scale σ, gradient norm bound C, total number of iterations T. Parallelization parameters: individual batch size B, number of workers R.

Initialize θ_0 randomly with the same seed for all R workers.

Partition training data into R pieces and distribute to each worker.

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

Take a random batch $L_{t,r}$ of size B on the local data for each worker r

Compute local gradient

For each $i \in L_{t,r}$, compute $\mathbf{g}_{t,r}(x_i) \leftarrow \nabla_{\theta_t}\mathcal{L}(\theta_t, x_i)$

Clip local gradient

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

Noise Addition

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

Gradient AllReduce

sum all $\tilde{\mathbf{g}}_{t,r}$ across all of the workers into $\tilde{\mathbf{g}}_t$.

Update $\theta_{t+1} \leftarrow \theta_t - \eta_t \tilde{\mathbf{g}}_t / R$ on all workers.

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

AllReduce = Reduction + Broadcast

Distributed DPSGD Training: implementation (V1)

DistributedDataParallel automatically handles Gradient Reduction

```
dp_device_ids = [local_rank]
device = torch.device('cuda', local_rank)
model = Network()
model.to(device)
model = torch.nn.parallel.DistributedDataParallel(model, device_ids=dp_device_ids)
```

DistributedSampler automatically handles Data Partition

```
train_sampler = torch.utils.data.distributed.DistributedSampler(trainset)
train_loader = torch.utils.data.DataLoader(trainset, batch_size=batch_size, sampler=tr
```

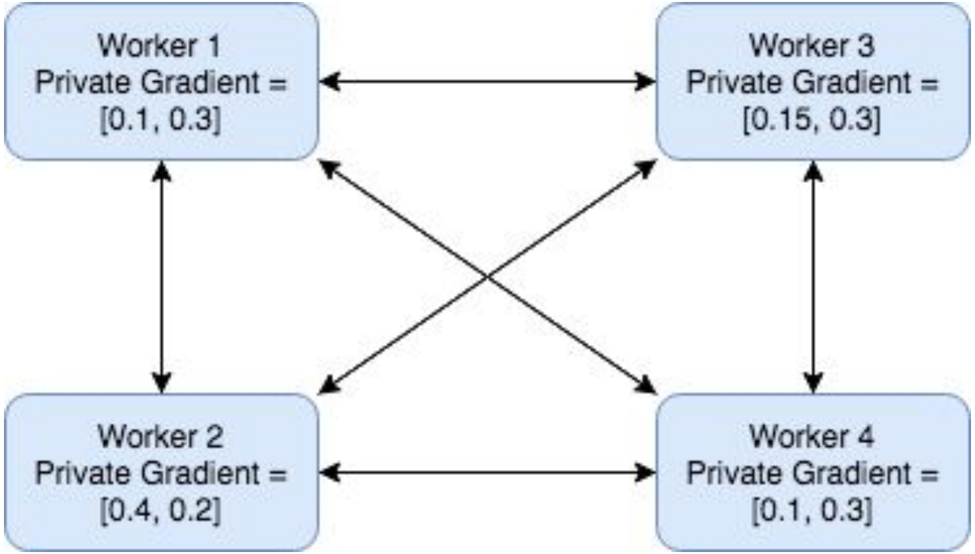
Distributed DPSGD Training: implementation (V2)

Implemented Data Partition and Gradient AllReduce from scratch, without using built-in library.

AllReduce Algorithm:

- Simple to implement; easy for debugging and analysis.
- Sending unnecessary messages, can be further improved.
- Obtained great performance in the experiment.

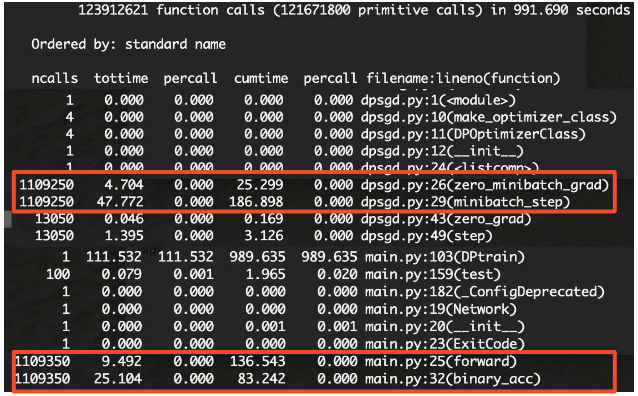
Message Passing Illustration:



Baseline Model: Result

1. bottleneck for optimization
    a. backwards propagation
    b. mini-batch in DPSGD

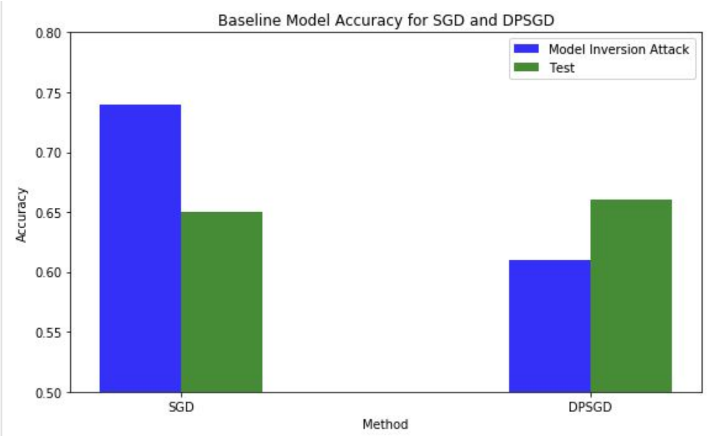
2. Model Accuracy:
    a. Test Accuracy:
    b. Model inversion attack[1] accuracy:



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

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.000    0.000    0.000    0.000 dpsgd.py:1(<module>)
        4    0.000    0.000    0.000    0.000 dpsgd.py:10(make_optimizer_class)
        4    0.000    0.000    0.000    0.000 dpsgd.py:11(DPOptimizerClass)
        1    0.000    0.000    0.000    0.000 dpsgd.py:12(__init__)
        1    0.000    0.000    0.000    0.000 dpsgd.py:24(<listcomp>)
  1109250    4.704    0.000   25.299    0.000 dpsgd.py:26(zero_minibatch_grad)
  1109250   47.772    0.000  186.898    0.000 dpsgd.py:29(minibatch_step)
    13050    0.046    0.000    0.169    0.000 dpsgd.py:43(zero_grad)
    13050    1.395    0.000    3.126    0.000 dpsgd.py:49(step)
        1  111.532  111.532  989.635  989.635 main.py:103(DPtrain)
      100    0.079    0.001    1.965    0.020 main.py:159(test)
        1    0.000    0.000    0.000    0.000 main.py:182(_ConfigDeprecated)
        1    0.000    0.000    0.000    0.000 main.py:19(Network)
        1    0.000    0.000    0.001    0.001 main.py:20(__init__)
        1    0.000    0.000    0.000    0.000 main.py:23(ExitCode)
  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)
```



1: Model inversion attack is a famous privacy attack against machine learning models. The access to a model is abused to infer information about the training data, which raised serious concerns given that training data usually contain privacy sensitive information. We use the success rate of model inversion attack as the criteria for the effectiveness of DP training.

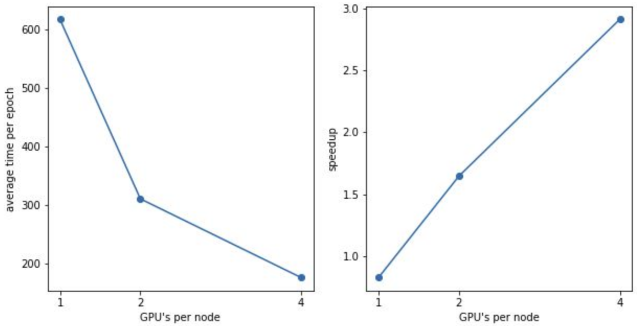
Distributed Training: Result

```
 ncalls  tottime  percall  cumtime  percall filename:lineno(function)
401520    1.866    0.000   27.993    0.000 dpsgd.py:26(zero_minibatch_grad)
401520   25.077    0.000  173.153    0.000 dpsgd.py:29(minibatch_step)
 19120    0.131    0.000    1.036    0.000 dpsgd.py:43(zero_grad)
 19120    3.424    0.000   10.424    0.001 dpsgd.py:49(step)
     1    0.000    0.000    0.000    0.000 dist_scratch_main.py:169(<listcomp>)
     1    0.000    0.000   27.018   27.018 dist_scratch_main.py:21(init_process)
401520   66.600    0.000  191.118    0.000 dist_scratch_main.py:29(average_gradients)
     1   45.675   45.675 1048.284 1048.284 dist_scratch_main.py:36(DPtrain)
```

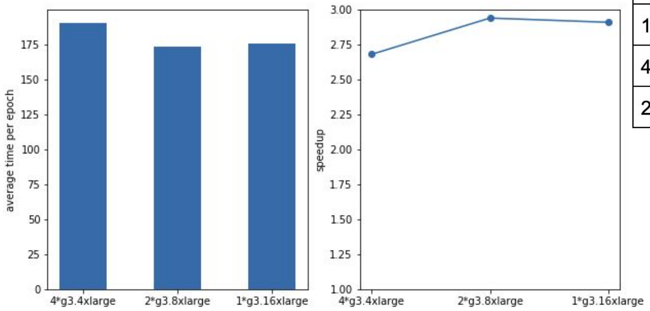
- Computing parallelism increases speedup
- Different distributions of GPU achieves approximately the same speedup
- intra-node communication VS inter-node communication overhead
- Economy-time tradeoff

Experiment	# GPU	Time (s)	price / hour	Cost
1*g3.4xlarge	1	6300	1.14	1.995
1*g3.8xlarge	2	3205	2.28	2.03
1*g3.16xlarge	4	1813	4.56	2.3
4*g3.4xlarge	4	2144	4.56	2.72
2*g3.8xlarge	4	1937	4.56	2.45

Package Version: Experiment with different number of GPU



Package Version: Experiment with different distributions of GPU



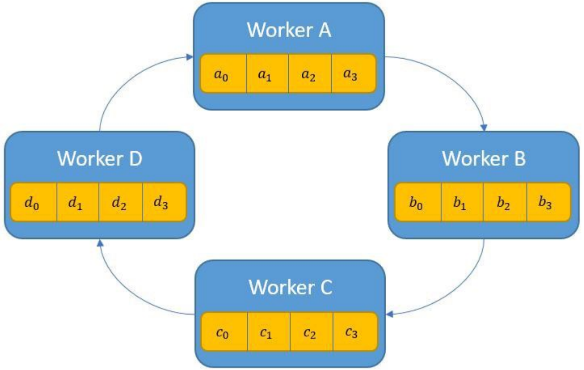
Discussion & Future Work

Discussion

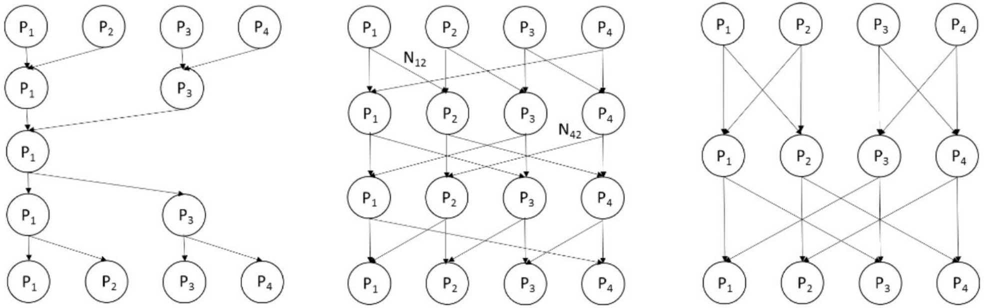
- Data Processing Speed-up: 21.89
- Distribute DPSGD Speed-up: 3.09

Future Work

- Implement advanced All-Reduce algorithms:
    - Tree All-Reduce
    - Round-robin All-Reduce
    - Butterfly All-Reduce
- Try to use Ring All-Reduce algorithm



Ring All-Reduce



(a) Tree AllReduce (b) Round-robin AllReduce (c) Butterfly AllReduce