Communication-efficient
Distributed SGD with Sketching

Nikita Ivkin*, Daniel Rothchild*, Enayat Ullah*, Vladimir Braverman, Ion Stoica, Raman Arora

* equal contribution
Going distributed: why?

- Large scale machine learning is moving to the distributed setting due to growing size of datasets/models, and modern learning paradigms like Federated learning.

![Performance vs Data](graph.png)

- Deep Neural Networks
- Medium Neural Networks
- Shallow Neural Networks
- Traditional Machine Learning
Going distributed: how?

- data
- model
- hybrid

most popular
Going distributed: how?

- sync
- parameter server
- batch 1
- batch 2
- batch m
- all-gather
- hybrid topology
Going distributed: how?

Synchronization with the parameter server:
Going distributed: how?

Synchronization with the parameter server:
- mini-batches distributed among workers
Going distributed: how?

Synchronization with the parameter server:

- mini-batches distributed among workers
- each worker makes forward-backward pass and computes the gradients
Going distributed: how?

Synchronization with the parameter server:

- mini-batches distributed among workers
- each worker makes forward-backward pass and computes the gradients
- workers send gradients to parameter server
Going distributed: how?

Synchronization with the parameter server:

- mini-batches distributed among workers
- each worker makes forward-backward pass and computes the gradients
- workers send gradients to parameter server

\[ g_1, g_2, \ldots, g_m \]
Going distributed: how?

Synchronization with the parameter server:

- mini-batches distributed among workers
- each worker makes forward-backward pass and computes the gradients
- workers send gradients to parameter server
- parameter server sums it up and sends it back to all workers

\[ G = g_1 + g_2 + \ldots + g_m \]
Going distributed: how?

Synchronization with the parameter server:

- mini-batches distributed among workers
- each worker makes forward-backward pass and computes the gradients
- workers send gradients to parameter server
- parameter server sums it up and sends it back to all workers
Going distributed: how?

Synchronization with the parameter server:

- mini-batches distributed among workers
- each worker makes forward-backward pass and computes the gradients
- workers send gradients to parameter server
- parameter server sums it up and sends it back to all workers
Going distributed: how?

Synchronization with the parameter server:

- mini-batches distributed among workers
- each worker makes forward-backward pass and computes the gradients
- workers send gradients to parameter server
- parameter server sums it up and sends it back to all workers
- each worker makes a step

\[
\begin{align*}
\text{worker 1:} & \quad w_t = w_{t-1} + \alpha G \\
\text{worker 2:} & \quad w_t = w_{t-1} + \alpha G \\
\text{worker m:} & \quad w_t = w_{t-1} + \alpha G \\
\end{align*}
\]
Going distributed: what’s the problem?

- Slow communication overwhelms local computations:
  - parameter vector for large models can weight up to 0.5 GB
  - synchronize every fraction of a second

```
  parameter
  vector
  for large
  models
  weight up
to 0.5 GB

  synchronize
every fraction
of a
second
```

```
worker 1  worker 2  worker m
```

```
batch 1  batch 2  batch m
```
Going distributed: what’s the problem?

- Slow communication overwhelms local computations:
  - parameter vector for large models can weight up to 0.5 GB
  - synchronize every fraction of a second

- Mini batch size has limit to its growth

Computation resources are wasted

Workers

Data

Batch 1

Batch 2

Batch m

Worker 1

Worker 2

Worker m
Going distributed: how others deal with it?

- Compressing the gradients:
  - Quantization

![Compressing the gradients diagram](image)

- Sparsification

![Sparsification diagram](image)
Quantization

- Quantizing gradients can give a constant factor decrease in communication cost.
- Simplest quantization to 16-bit, but all the way to 2-bit (TernGrad [1]) and 1-bit (SignSGD [2]) have been successful.
- Quantization techniques can in principle be combined with gradient sparsification.

References:
Sparsification

● Existing techniques either communicate $\Omega(Wd)$ in the worst case, or are heuristics; $W$ - number of workers, $d$ - dimension of gradient.

● [1] showed that SGD (on 1 machine) with top-$k$ gradient updates and error accumulation has desirable convergence properties.

● Q. Can we extend the top-$k$ to the distributed setting?
  ○ MEM-SGD [1] (for 1 machine, extension to distributed setting is sequential)
  ○ top-k SGD [2] (assumes that global top $k$ is close to sum of local top $k$)

● We resolve the above using sketches!

Streaming model

Stream: \( p_1, p_2, \ldots, p_m \in [n] \)

Goal: find “frequent” items (icebergs, elephants, heavy hitters)

Restrictions:
- access the data sequentially, make only one pass
- small space, fast updates, fast query

Frequency: \( f_i = \# \{ j \in [m]: p_j = i \} \)

\((\alpha, \ell_p)\) heavy hitter: \( f_i \geq \alpha \| f \|_p \)

Example:

\[
\begin{array}{cccccccccccccccc}
\text{vector} & \text{index} & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\
\hline
\text{value} & i & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\
\hline
\end{array}
\]

- \( f \) = frequency vector

- \( \| f \|_1 = \sum f_i = 17 \)

- \( f_5 \geq 0.3 \| f \|_1 \)

- 5 is \((0.3, \ell_1)\) heavy hitter

- \( \| f \|_2 = (\sum f_i^2)^{1/2} = 17 \)

- \( f_5 \geq 0.8 \| f \|_2 \)

- 5 is \((0.8, \ell_2)\) heavy hitter

\( \| \cdot \|_2 < \| \cdot \|_1 \) \Rightarrow finding \( \ell_2 \) heavy hitters is more challenging than \( \ell_1 \)

Approximate answer with high probability is OK
$\ell_2$ norm estimation (AMS’96)

Stream: $p_1, p_2, \ldots, p_m \in [n]$

Want to find: $\| f \|_2$

<table>
<thead>
<tr>
<th>$i$</th>
<th>$f_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>
| 3   |      | frequencies of balls
$\ell_2$ norm estimation (AMS’96)

<table>
<thead>
<tr>
<th>$i$</th>
<th>$Z_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>+1</td>
</tr>
<tr>
<td>1</td>
<td>+1</td>
</tr>
<tr>
<td>2</td>
<td>+1</td>
</tr>
<tr>
<td>3</td>
<td>+1</td>
</tr>
<tr>
<td>4</td>
<td>+1</td>
</tr>
<tr>
<td>5</td>
<td>+1</td>
</tr>
<tr>
<td>6</td>
<td>+1</td>
</tr>
<tr>
<td>7</td>
<td>+1</td>
</tr>
<tr>
<td>8</td>
<td>+1</td>
</tr>
<tr>
<td>9</td>
<td>+1</td>
</tr>
<tr>
<td>10</td>
<td>+1</td>
</tr>
<tr>
<td>11</td>
<td>+1</td>
</tr>
<tr>
<td>12</td>
<td>+1</td>
</tr>
<tr>
<td>13</td>
<td>-1</td>
</tr>
<tr>
<td>14</td>
<td>-1</td>
</tr>
<tr>
<td>15</td>
<td>-1</td>
</tr>
<tr>
<td>16</td>
<td>-1</td>
</tr>
<tr>
<td>17</td>
<td>-1</td>
</tr>
<tr>
<td>18</td>
<td>-1</td>
</tr>
<tr>
<td>19</td>
<td>-1</td>
</tr>
<tr>
<td>20</td>
<td>-1</td>
</tr>
</tbody>
</table>

**Algorithm 1** AMS sketch

1: $X = 0$
2: function **UPDATE**($i$)
3: $X += Z_i$
4: return $X^2$

Stream: $p_1, p_2, \ldots, p_m \in [n]$

$X = \sum_{j=1}^{m} Z_{p_j}$

$X^2 = 144$

$\|f\|_2^2 = 150$
\( \ell_2 \) norm estimation (AMS’96)

Stream: \( p_1, p_2, \ldots, p_m \in [n] \)

\[
X = \sum_{j=1}^{m} Z_{p_j} = \sum_{i=1}^{n} Z_i f_i
\]

\[
E(X^2) = E\left( \sum_{i=1}^{n} Z_i^2 f_i^2 \right) + E\left( \sum_{i \neq j} f_i f_j Z_i Z_j \right) = \sum_{i=1}^{n} f_i^2 = \|f\|_2^2
\]

\( Z_i^2 = 1 \)

\[
\sum_{i=1}^{n} f_i^2
\]

\[
\sum_{i=1}^{n} f_i f_j E(Z_i) E(Z_j) = 0
\]

(\( E(Z_i) = 0 \) unless \( Z_i \) is indep)
Count Sketch

\[
\begin{align*}
(1, f_1) & \\
(2, f_2) & \\
(i, f_i) & \\
(d, f_d) & \\
\end{align*}
\]

coordinate updates

\[
\begin{align*}
sign hash & \rightarrow +1 \\
bucket hash & \rightarrow 7 \\
 & \rightarrow +f_i
\end{align*}
\]
Count Sketch

coordinate updates

bucket hashes: $i \rightarrow \{1, \ldots, b\}$

sign hashes: $i \rightarrow \pm 1$
Mergebility

\[ S_M = \sum S_i = \sum A g_i = S(\sum g_i) \]

\[ S_M \text{ recovers top } k \text{ of } \sum g_i \]

\[ S_1 \]
\[ S_2 \]
\[ S_W \]

\[ S_1 = S(g_1) \]
\[ S_2 = S(g_2) = A g_2 \]
\[ S_W = S(g_W) \]
Compression scheme

Synchronization with the parameter server:

workers

data

worker 1

batch 1

worker 2

batch 2

worker m

batch m

parameter server
Compression scheme

Synchronization with the parameter server:
- mini-batches distributed among workers
Compression scheme

Synchronization with the parameter server:
- mini-batches distributed among workers
- each worker makes forward-backward pass and computes and sketch the gradients
Compression scheme

Synchronization with the parameter server:
- mini-batches distributed among workers
- each worker makes forward-backward pass and computes and sketch the gradients

$S(g_1)$  $S(g_2)$  ...  $S(g_m)$
Compression scheme

Synchronization with the parameter server:
- mini-batches distributed among workers
- each worker makes forward-backward pass and computes and sketch the gradients
- workers send sketches to parameter server

\[ S(\mathbf{g}_1) \]
\[ S(\mathbf{g}_2) \]
\[ S(\mathbf{g}_m) \]
Compression scheme

Synchronization with the parameter server:

- mini-batches distributed among workers
- each worker makes forward-backward pass and computes and sketch the gradients
- workers send sketches to parameter server

\[ S_1, S_2, \ldots, S_m \]
Compression scheme

Synchronization with the parameter server:

- mini-batches distributed among workers
- each worker makes forward-backward pass and computes and sketch the gradients
- workers send sketches to parameter server
- parameter server merge the sketches, extract top k and send it back

\[ S = S_1 + S_2 + \ldots + S_m \]
Compression scheme

Synchronization with the parameter server:

- mini-batches distributed among workers
- each worker makes forward-backward pass and computes and sketch the gradients
- workers send sketches to parameter server
- parameter server merge the sketches, extract top k and send it back

\[ G' = \text{topk}(S) \]
Compression scheme

Synchronization with the parameter server:

- mini-batches distributed among workers
- each worker makes forward-backward pass and computes and sketch the gradients
- workers send sketches to parameter server
- parameter server merge the sketches, extract top k and send it back
Compression scheme

Synchronization with the parameter server:

- mini-batches distributed among workers
- each worker makes forward-backward pass and computes and sketch the gradients
- workers send sketches to parameter server
- parameter server merge the sketches, extract top k and send it back
- each worker makes a step

$$w_t = w_{t-1} + \alpha G$$
Algorithm and theory

Algorithm 2 SKETCHED-D-SGD

Input: $k, \epsilon, \xi, \delta, W, P$

1: $\eta_t \leftarrow \frac{1}{\xi + \xi}, q_t \leftarrow (\xi + t)^2, Q_T = \sum_{t=1}^{T} q_t, a_0 = 0$
2: while $t = 1, 2, \ldots T$ do
3:   Compute stochastic gradient $g_t^i$ (Worker$_i$)
4:   Error correction: $\tilde{g}_t^i = \eta_t g_t^i + a_{t-1}^i$ (Worker$_i$)
5:   Compute sketches $S_t^i$ of $\tilde{g}_t^i$ (Worker$_i$)
6:   Communicate sketches $S_t^i$ to master (Worker$_i$)
7:   Aggregate sketches $S_t = \frac{1}{W} \sum_{i=1}^{W} S_t^i$ (Master)
8:   Unsketch: Get top $Pk$ elements from $S_t$ (Master)
9:   Communicate co-ordinates of $Pk$ elements to all workers and get exact values of top $k$ as $\tilde{g}_t$ (Master)
10: Communicate $\tilde{g}_t$ to all workers (Master)
11: Update $w_{t+1} = w_t - \tilde{g}_t$ (Master)
12: Communicate the $k$ updated model parameters of $w_{t+1}$ to all workers (Master)
13: Error accumulation: $a_t^i = \tilde{g}_t^i - \tilde{g}_t$ (Worker$_i$)
14: end while
Output: $\hat{w}_T = \frac{1}{Q_T} \sum_{t=1}^{T} q_t w_t$

- **Theoretical guarantees**
  - Converges at $O(1/WT)$ rate, at par with SGD for smooth strongly convex functions, where $W$ is the number of workers.
  - Communicates $O(k \log^2 d)$, size of sketch, $0 < k < d$, $d$: dimension of model.

- **Scalability**
  - More workers - Increasing the number of workers $W$ increases the rate of convergence (suitable for Federated learning)
  - Bigger models - Increasing the model size $d$ increases the compression ratio $d/k \log^2 d$. 
Empirical Results

<table>
<thead>
<tr>
<th></th>
<th>Vanilla distributed SGD</th>
<th>BLEU (transformer)</th>
<th>BLEU (LSTM)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Top-100,000 SGD</td>
<td>26.65</td>
<td>22.2</td>
<td></td>
</tr>
<tr>
<td>SKETCHED-SGD, 20x compression</td>
<td>26.87¹</td>
<td>22.2</td>
<td></td>
</tr>
<tr>
<td>SKETCHED-SGD, 40x compression</td>
<td>26.79²</td>
<td>20.95³</td>
<td></td>
</tr>
</tbody>
</table>

BLEU scores on the test data achieved for vanilla distributed SGD, top-k SGD, and SKETCHED-SGD with 20x and 40x compression. Larger BLEU score is better.

Figure 1: Learning curves for a transformer model trained on the WMT 2014 English to German translation task. All models included here achieve comparable BLEU scores after 60,000 iterations (see Table 1). Each run used 4 workers.
Empirical Results

Comparison between SKETCHED-SGD and local top-k SGD on CIFAR10. The best overall compression that local top-k can achieve for many workers is 2x.
Computational overhead

Simple to parallelize the sketching part:

```python
for each coordinate:
    for each row:
        compute hashes (bucket + sign)
        update corresponding counter
```

100x acceleration on modern GPU

Specifics of distributed SGD application:
- gradient vector is already on GPU
- for reasonable $d$, all hashes can be precomputed
- one-liner to parallelize using pytorch framework (20x speed up)

```python
table[row,:] += torch.bincount(bucketsHashes[row,:], signsHashes[row]*vec)
```
Thanks a lot!