Steps Towards Collaborative & Decentralized ML

Martin Jaggi
EPFL
Machine Learning and Optimization Laboratory
mlo.epfl.ch
Collaborative & Distributed Training
Big Picture
- Status Quo

AI utility

data

personal data

device

device

device

device

device
Federated Learning
Decentralized Learning

Local Data

Model Updates

Data
Motivation

Applications:
any ML system with user data
servers, devices, sensors, hospitals, ...

Advantages:
AI utility, control and privacy
aligned with data ownership
Required Building Blocks

Decentralized ML

Efficiency

Privacy

Robustness
Required Building Blocks

- **Efficiency: Communication & Compute**
  - on-device learning, Edge AI
  - peer-to-peer communication

- **Privacy**
  - data locality, leakage?, attacks?

- **Robustness & Incentives**
  - tolerate bad players, reward collaboration
Stochastic Gradient Descent (SGD)

\[
\min_x f(x) = \frac{1}{|\text{data}|} \sum_{i \in \text{data}} f_i(x)
\]

\[i_t \sim \text{Uniform}(1, |\text{data}|)\]

\[x_{t+1} := x_t - \gamma_t \nabla f_{i_t}(x_t)\]
## Communication Efficiency

A compressed version of model updates?

### Examples:

<table>
<thead>
<tr>
<th>Example</th>
<th>Communication Reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>quantization (e.g. 1-bit SGD)</td>
<td>32x</td>
</tr>
<tr>
<td>top (k=1%) of all the entries</td>
<td>100x</td>
</tr>
<tr>
<td>rank-1 approximation</td>
<td>&gt;100x</td>
</tr>
</tbody>
</table>
Gradient Compression

A compressed version of model updates?

Input neurons

Output neurons

Layer gradient

Sign

Sign + Norm

Top $K$

Random $K$

Random Block
SGD fails with naive/biased compressors.

\[
\min_{x \in \mathbb{R}^2} |x_1 + x_2| + 2|x_1 - x_2|
\]
Error Feedback

**PowerSGD**: Practical Low-Rank Gradient Compression for Distributed Optimization

Rapid Low-rank Approximation

PowerSGD sees a layer's gradient as a matrix. It approximates this matrix as the product of two narrow matrices by using one step of power iteration. This approximation is coarse, but only involves two multiplications of the gradient matrix and a very narrow one, followed by an orthogonalization of the output. This is much faster than an SGD.

PowerSGD converges, even with this coarse approximation. This is mainly due to the error feedback mechanism.

**Input features**

**Output features**

**Error Feedback**

Though PowerSGD compression is biased and of low quality, the algorithm can converge in a similar number of steps as full-precision SGD. This is thanks to error feedback.

**ResNet-50 on Cifar-10**

In normal, uncompressed, SGD, the workers average their gradients after each iteration. This average can be computed efficiently with hierarchical all-reduce communication.

Unfortunately, compressed algorithms cannot hierarchically average their compressed gradients. Therefore, these algorithms resort to less scalable all-to-all communication or a parameter server.

The power iteration step of PowerSGD, effectively multiplies the average gradient matrix across workers with the same narrow matrix. Due to linearity, this operation is equivalent to averaging the small output matrices.

Because all communication in PowerSGD is just a merge operation, it enjoys all the benefits of all-reduce.

**Scalability**

Due to its fast compression algorithm and strong reduction in communication around 10x in our experiments, PowerSGD scales well on slow backends, but can still improve over SGD when using Nvidia's highly optimized *CCL.*

**Loss Gradient Compression**

In distributed training, workers typically exchange their mini-batch gradients at every iteration. These gradients can be (200s of megabytes) large, so this communication limits the scalability of distributed optimization.

Loss compression of gradients before sharing them across workers is a popular approach to mitigate this problem.

**Code**

Download the code at [GitHub](https://github.com/epfml/powersgd).

In normal, uncompressed, SGD, the workers average their gradients after each iteration. This average can be computed efficiently with hierarchical all-reduce communication.

Unfortunately, compressed algorithms cannot hierarchically average their compressed gradients. Therefore, these algorithms resort to less scalable all-to-all communication or a parameter server.

The power iteration step of PowerSGD, effectively multiplies the average gradient matrix across workers with the same narrow matrix. Due to linearity, this operation is equivalent to averaging the small output matrices.

Because all communication in PowerSGD is just a merge operation, it enjoys all the benefits of all-reduce.

**Scalability**

Due to its fast compression algorithm and strong reduction in communication around 10x in our experiments, PowerSGD scales well on slow backends, but can still improve over SGD when using Nvidia's highly optimized *CCL.*

**ResNet-50 on Cifar-10**

In our experiments, PowerSGD can be used plug-and-play with an existing optimizer without re-tuning the optimizer's hyperparameters. With a high enough compression rate, PowerSGD can achieve the same test accuracy as uncompressed, full-precision SGD while enjoying reductions in communication of more than 10x.

**1h2s !o#els**

Sai Praneeth Sarimireddy, Martin Jadaan

**Plu**

**Pla**

In our experiments, PowerSGD can be used plug-and-play with an existing optimizer without re-tuning the optimizer's hyperparameters. With a high enough compression rate, PowerSGD can achieve the same test accuracy as uncompressed, full-precision SGD while enjoying reductions in communication of more than 10x.
PowerSGD: Practical Low-Rank Gradient Compression for Distributed Optimization

PowerSGD sees a layer’s gradient as a matrix. It approximates this matrix as the product of two narrow matrices by using one step of power iteration. This approximation is coarse, but only involves two multiplications of the gradient matrix and a very narrow one, followed by an orthogonalization of the output. This is much faster than an SVD.

PowerSGD converges, even with this coarse approximation. This is mainly due to the error feedback mechanism.

In normal, uncompressed, SGD, the workers average their gradients after each iteration. This average can be computed efficiently with hierarchical all-reduce communication. Unfortunately, compressed algorithms cannot hierarchically average their compressed gradients. Therefore, these algorithms resort to less scalable all-to-all communication or a parameterserver.

The power iteration step of PowerSGD, effectively multiplies the average gradient matrix across workers with the same narrow matrix. Due to linearity, this operation is equivalent to averaging the small output matrices.

Because all communication in PowerSGD is just an algebraic operation, it enjoys all the benefits of all-reduce.

In our experiments, PowerSGD can be used plug-and-play with an existing optimizer without re-tuning the optimizer’s hyperparameters. With a high enough compression ratio, PowerSGD can achieve the same test accuracy as uncompressed, full-precision SGD while enjoying reductions in communication of more than 10x.

Loss' Gradient Compression

In distributed training, workers typically exchange their mini-batch gradients at every iteration. These gradients can be dozens of megabytes large, so this communication limits the scalability of distributed optimization.

Loss’ compression of gradients before sharing them across workers is a popular approach to mitigate this problem.

Error Feedback

\[
\begin{align*}
\text{compressed} & \quad \theta_i \\
\text{error} & \quad \text{gradient} \\
\theta_i & \quad \theta_i+1
\end{align*}
\]
PowerSGD: Practical Low-Rank Gradient Compression for Distributed Optimization

Rapid Low-rank Approximation

PowerSGD sees a layer’s gradient as a matrix. It approximates this matrix as the product of two narrow matrices by using one step of power iteration. This approximation is coarse, but only involves two multiplications of the gradient matrix and a very narrow one, followed by an orthogonalization of the output. This is much faster than an SGD.

PowerSGD converges, even with this coarse approximation. This is mainly due to the error feedback mechanism.

$\text{Input features}$ $\rightarrow$ $\text{Output features}$

Though PowerSGD compression is biased and of low quality, the algorithm can converge in a similar number of steps as full-precision SGD. This is thanks to error feedback.

Seide et al. (2013), Stich et al. (2018), Sarimireddy et al. (2019).

All-reduce Communication

In normal, uncompressed, SGD, the workers average their gradients after each iteration. This average can be computed efficiently with hierarchical all-reduce communication.

Unfortunately, compressed algorithms cannot hierarchically average their compressed gradients. Therefore, these algorithms resort to less scalable all-to-all communication or a parameter server.

The power iteration step of PowerSGD effectively multiplies the average gradient matrix across workers with the same narrow matrix. Due to linearity, this operation is equivalent to averaging the small output matrices.

Because all communication in PowerSGD is just an elementwise operation, it enjoys all the benefits of all-reduce.

Scalability

Due to its fast compression algorithm and strong reduction in communication (x in our experiments), PowerSGD scales well on slow backends, but can still improve over SGD when using Nvidia’s highly optimized *CCL.

ResNet-18 on Cifar-10 (784 or workers) (x)

Plugging PowerSGD into an existing optimizer without re-tuning the optimizer’s hyperparameters. With a high enough compression ratio, PowerSGD can achieve the same test accuracy as uncompressed, full-precision SGD while enjoying reductions in communication of more than x.

In our experiments, PowerSGD can be used plug-and-play with an existing optimizer without re-tuning the optimizer’s hyperparameters. With a high enough compression ratio, PowerSGD can achieve the same test accuracy as uncompressed, full-precision SGD while enjoying reductions in communication of more than x.
Error Feedback: Convergence Rate

\[ \delta: \text{compression ratio} \]
\[ \| C(x) - x \|_2^2 \leq (1 - \delta) \| x \|_2^2 \]

SGD on smooth non-convex objectives (w/ central coordinator)

\[ \mathbb{E} \| \nabla f(\bar{x}_t) \|_2^2 \leq \Theta \left( \frac{1}{\sqrt{nT}} + \frac{1}{\delta^2 T} \right) \]

Error Feedback Fixes SignSGD and other Gradient Compression Schemes
Gradient Compression

Input neurons

Output neurons

Layer gradient

Sign
Sign + Norm
Top $K$
Random $K$
Random Block
Low-rank (ours)
## PowerSGD for NN training

<table>
<thead>
<tr>
<th>Compression Level</th>
<th>Test Accuracy</th>
<th>Sent/epoch</th>
<th>All-reduce</th>
<th>Time/batch</th>
</tr>
</thead>
<tbody>
<tr>
<td>No compression</td>
<td>94.3%</td>
<td>1023 MB</td>
<td></td>
<td>312 ms</td>
</tr>
<tr>
<td><strong>Medium</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rank 7</td>
<td>94.6%</td>
<td>24 MB</td>
<td>✓</td>
<td>285 ms</td>
</tr>
<tr>
<td>Random Block</td>
<td>93.3%</td>
<td>24 MB</td>
<td>✓</td>
<td>243 ms</td>
</tr>
<tr>
<td>Random K</td>
<td>94.0%</td>
<td>24 MB</td>
<td>✓</td>
<td>540 ms</td>
</tr>
<tr>
<td>Sign+Norm</td>
<td>93.9%</td>
<td>32 MB</td>
<td>✗</td>
<td>429 ms</td>
</tr>
<tr>
<td>Top K</td>
<td>94.4%</td>
<td>32 MB</td>
<td>✗</td>
<td>444 ms</td>
</tr>
<tr>
<td><strong>High</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rank 2</td>
<td>94.4%</td>
<td>8 MB</td>
<td>✓</td>
<td>239 ms</td>
</tr>
<tr>
<td>Random Block</td>
<td>87.8%</td>
<td>8 MB</td>
<td>✓</td>
<td>240 ms</td>
</tr>
<tr>
<td>Random K</td>
<td>92.6%</td>
<td>8 MB</td>
<td>✓</td>
<td>534 ms</td>
</tr>
<tr>
<td>Top K</td>
<td>93.6%</td>
<td>8 MB</td>
<td>✗</td>
<td>411 ms</td>
</tr>
</tbody>
</table>
lazy compression: power iteration is a linear operation!

- non-i.i.d. data
- suitable for all-reduce
Model Compression with Error Feedback

- Prune most weights (set to zero)
- Set to limited precision
- Interactive while training
Consensus

\[ x_{t+1}^i := \frac{1}{\text{deg}_i} \sum_{j:\text{neighbours}} x_t^j \]
Consensus with Compression

Naive solution:

\[ x_t^i := \frac{1}{deg_i} \sum_{j: \text{neighbours}} Q(x_j^i) \]
Error Compensation

Our method:

\[
\Delta_i^t := x_i^t - \hat{x}_i^t \\
\hat{x}_i^{t+1} := \hat{x}_i^t + Q(\Delta_i^t) \\
x_{i+1}^t := x_i^t + \gamma \frac{1}{\text{deg}_i} \sum_{j:\text{neighbours}} (\hat{x}_j^{t+1} - \hat{x}_i^{t+1})
\]
Choco-SGD

SGD step:

\[ x_{t+\frac{1}{2}}^i := x_t^i - \gamma_t \nabla f_j(x_t^j) \]

\[ x_{t+1}^i := \text{consensus}_\text{with}_\text{compression} \left( x_{t+\frac{1}{2}}^j \right) \]
Convergence (Non-Convex Case)

\[
\frac{1}{T+1} \sum_{t=0}^{T} \| \nabla f(\bar{x}_t) \|^2 = \mathcal{O} \left( \frac{1}{\sqrt{nT}} + \frac{n}{\delta^2 \rho^4 T} \right)
\]

\( \delta \) — compression ratio  \( \delta \in [0,1] \),  \( \delta = 1 \) for no compression

\( \rho \) — spectral gap of the graph topology

* linear speedup in the number of workers
Decentralized DL

Resnet20 on Cifar 10

Language model (3-layer LSTM) on WikiText-2

Social Network Topology, 32 nodes of max deg 14
Sign quantization
DL in Datacenter

Resnet50 on ImageNet-1k
Ring of 8 nodes, each has 4 P100 GPUs
Conclusions - Choco

- First **consensus algorithm** that converges linearly with arbitrary compression
- First **decentralized SGD** algorithm that converges with arbitrary compression
- **Practical performance**
Building Blocks for Decentralized ML

• **Efficiency: Communication & Compute**
  on-device learning, Edge AI
  peer-to-peer communication

• **Privacy**
  data locality, leakage?, attacks?

• **Robustness & Incentives**
  tolerate bad players, reward collaboration
Open Source Project:

**MLbench - Distributed Machine Learning Benchmark**

Public and reproducible reference implementations and benchmarks for distributed machine learning algorithms, frameworks and systems.

[mlbench.github.io](https://mlbench.github.io)
References

- Error Feedback Fixes SignSGD and other Gradient Compression Schemes
  - [github.com/epfml/error-feedback-SGD](https://github.com/epfml/error-feedback-SGD)

- PowerSGD: Practical Low-Rank Gradient Compression for Distributed Optimization
  - [github.com/epfml/powerSGD](https://github.com/epfml/powerSGD)

- Decentralized Stochastic Optimization and Gossip Algorithms with Compressed Communication

- Decentralized Deep Learning with Arbitrary Communication Compression
  - [github.com/epfml/ChocoSGD](https://github.com/epfml/ChocoSGD)
Thanks

Sebastian U. Stich, Sai Praneeth Karimireddy, Anastasia Koloskova, Thijs Vogels, Tao Lin, Quentin Rebjock

Machine Learning and Optimization Laboratory

website: mlo.epfl.ch
Supplementary Slides
Optimization: Assumptions

- finite sum objective

\[
\min_{\mathbf{x} \in \mathbb{R}^d} \left[ f(\mathbf{x}) := \frac{1}{n} \sum_{i=1}^{n} f_i(\mathbf{x}) \right], \quad f_i := E_{\xi_i \sim D_i}[F_i(\mathbf{x}, \xi_i)], \quad \forall i \in [n]
\]

- smoothness of each \( f_i \)

- bounded variance (and 2nd moment) per worker

\[
E_{\xi_i} \| \nabla F_i(\mathbf{x}, \xi_i) - \nabla f_i(\mathbf{x}) \|^2 \leq \sigma_i^2, \quad E_{\xi_i} \| \nabla F_i(\mathbf{x}, \xi_i) \|^2 \leq G^2, \quad \forall \mathbf{x} \in \mathbb{R}^d, i \in [n]
\]