Distributed Machine Learning: The Good, the Bad, and the Hyperparameters

Dan Alistarh
IST Austria & NeuralMagic, Inc.
The Machine Learning “Cambrian Explosion”

Even if progress stalls, **ML is here to stay:** existing technologies already have significant industry adoption.
Distributed/parallel computing is the key enabler of computational speedups.
Distribution is Key

Training Deep Neural Networks Efficiently

• Large Datasets:
  • ImageNet: 1.3 Million images
    Google OpenImages: 9 Million images
  • NIST2000 Switchboard dataset: 2000 hours
    Proprietary speech datasets: > 30,000 hours (3.5 years)
  ➢ Distributed training is necessary

• Large Models:
  • ResNet-152 [He et al. 2015]: 152 layers, 60 million parameters
  • LACEA [Yu et al. 2016]: 22 layers, 65 million parameters

Is efficient distributed machine learning a solved problem?
The Scalability Problem

CSCS: Europe’s Top Supercomputer (World 4th)
- 4500+ GPU Nodes, state-of-the-art interconnect

Task:
- Image Classification (ResNet-152 on ImageNet)
- Single Node time (TensorFlow): 19 days
- 1024 Nodes: 25 minutes (in theory)
The Scalability Problem

CSCS: Europe’s Top Supercomputer (World 4th)
- 4500+ GPU Nodes, state-of-the-art interconnect

Task:
- Image Classification (ResNet-152 on ImageNet)
- Single Node time (TensorFlow): 19 days
- 1024 Nodes: 25 minutes (in theory)
The Scalability Problem

CSCS: Europe’s Top Supercomputer (World 4th)
- 4500+ GPU Nodes, state-of-the-art interconnect

Task:
- Image Classification (ResNet-152 on ImageNet)
- Single Node time (TensorFlow): 19 days
- 1024 Nodes: 25 minutes (in theory)

Efficient distribution is still a non-trivial challenge for machine learning applications.
The Algorithm: Parallel Stochastic Gradient Descent

Synchronous Message-Passing System

- \( n \) nodes, fully-connected communication topology
Parallel SGD (large models)

Synchronous Message-Passing System
  - n nodes, fully-connected communication topology
Parallel SGD *(really large models)*

Synchronous Message-Passing System

- n nodes, fully-connected communication topology

![Diagram showing the process of parallel SGD, with nodes computing updates, averaging updates, and updating the model in rounds.](image-url)
Today’s Talk

Communication-Efficient Algorithms for Scalable Machine Learning

Quantization ➔ Sparsification ➔ Efficient Aggregation

Trade-offs: compression vs convergence vs parametrization.

ScaleML: An open-source framework implementing these techniques

Overview & Open Problems
The General Setting

Given:

• $n$ nodes, synchronous message-passing, fully-connected topology
• Dataset $D$: node $p_i$ is assigned dataset partition $D_i$
• Loss function $\text{Loss}(x, e) = \text{how \text{ "good" is the prediction of model } x \text{ on example } e}$

Wanted:

model $\mathbf{x}$ minimizing $f(\mathbf{x}) = f_1(\mathbf{x}) + f_2(\mathbf{x})$

\[ f_1(x) = E_{e \text{ from } D_1} \left[ \text{Loss}(x, e) \right] \]
\[ f_2(x) = E_{e \text{ from } D_2} \left[ \text{Loss}(x, e) \right] \]
The Algorithm: Data-Parallel Stochastic Gradient Descent

- Each node maintains a copy of the “model/parameter” $x$
- In each iteration $t$, until convergence:
  - Each node $i$ selects a sample $e_i$ uniformly at random from $D_i$
  - It computes the update $\nabla_t^i = \text{the gradient of } x_t \text{ at } e_i \text{ w.r.t. the Loss}$
  - Nodes average their updates: $\nabla_t = (\nabla_t^1 + \nabla_t^2)/2$
  - Update model: $x_{t+1} = x_t - \eta_t \nabla_t$, where $\eta_t$ is the learning rate.
Example: Distributed Mean Estimation

• Given distribution $D$, find a parameter $x \in \mathbb{R}^d$ which minimizes $\mathbb{E}_{e \in D} \left( ||x - e||^2 \right)$.

• In each iteration $t$ until convergence:
  - Each node $i$ selects a sample $e_i$ uniformly at random from its local set
  - It computes the gradient of its estimate $\nabla_t^i = e_i - x_t$
  - Nodes average their gradients to obtain $\nabla_t = (e_1 + e_2)/2 - x_t$, and update their estimates by $x_{t+1} = x_t - \eta_t \nabla_t$.

The SGD algorithm remains roughly the same whether we are optimizing complex neural networks or solving classic regression.

Why does averaging / parallelism help?

Intuition: two random samples are better than one!
A Bit of Theory (1)

• Assume we wish to minimize a differentiable function $f: \mathbb{R}^d \rightarrow \mathbb{R}$

• We apply the classic SGD iteration

$$x_{t+1} = x_t - \eta_t \nabla_t(x_t), \text{ where } E_{e \in D}[\nabla_t(x_t)] = \nabla f(x_t).$$

• Let $E[||\nabla_t(x) - \nabla f(x)||^2] \leq \sigma^2$ (variance bound)

**Theorem** [e.g. Bubeck15]: Given $f$ convex and smooth, and $R^2 = ||x_0 - x^*||^2$.

If we run SGD for $T = \mathcal{O}(R^2 \frac{2\sigma^2}{\varepsilon^2})$ iterations, then

$$E \left[ f\left(\frac{1}{T} \sum_{t=0}^{T} x_t \right) \right] - f(x^*) \leq \varepsilon.$$
A Bit of Theory (2)

• Assume we wish to minimize a differentiable function $f : \mathbb{R}^d \rightarrow \mathbb{R}$
• We apply the classic SGD iteration

$$x_{t+1} = x_t - \eta_t \nabla_t(x_t), \text{ where } E_{e \in D}[\nabla_t(x_t)] = \nabla f(x_t).$$

• Assume we are **averaging** over $P$ gradient estimators. Then $E[||\nabla_t(x) - \nabla f(x)||^2] \leq \sigma^2/P$.

**Theorem** [e.g. Bubeck15]: Given $f$ convex and smooth, and $R^2 = ||x_0 - x^*||^2$. If we run SGD for $T = \mathcal{O}(R^2 \frac{2\sigma^2}{P\varepsilon^2})$ iterations, then

$$E \left[ f\left(\frac{1}{T} \sum_{t=0}^{T} x_t \right) \right] - f(x^*) \leq \varepsilon.$$
Today’s Talk

Communication-Efficient Algorithms for Scalable Machine Learning

Method 1: Gradient Quantization
1BitSGD Quantization
[Microsoft Research, Seide et al. 2014]

Quantization function

\[ Q_i(v) = \begin{cases} 
    \text{avg}_+ & \text{if } v_i \geq 0, \\
    \text{avg}_- & \text{otherwise}
\end{cases} \]

where \( \text{avg}_+ = \text{mean}([v_i \text{ for } i: v_i \geq 0]) \), \( \text{avg}_- = \text{mean}([v_i \text{ for } i: v_i < 0]) \)

Accumulate the quantization error locally, and apply to the next update!

Accumulate the quantization error locally, and apply to the next update!

Compression rate \( \approx 32x \)

Does not always converge!

Seide et al (2014) “1-Bit Stochastic Gradient Descent and its Application to Data-Parallel Distributed Training of Speech DNNs”
Stochastic Quantization
[Alistarh, Grubic, Li, Tomioka, Vojnovic, NeurIPS17]

Quantization function

$$Q[v; s] = \|v\|_2 \cdot \text{sgn}(v_i) \cdot \xi_i(v, s)$$

The random variable $\xi_i$ encodes the integer quantization level for $v_i$.

Compression ratio $\approx 32/(\log s + 1)$
QSGD Properties

Quantization function

\[ Q[v_i; s] = \|v\|_2 \cdot \text{sgn}(v_i) \cdot \xi_i(v, s) \]

- Properties

1. **Unbiasedness**
   \[ E[Q[v_i; s]] = v_i \]
   \(\rightarrow\) ensures convergence since \( E[Q[V_t(x_t)]] = Vf(x_t) \).

2. **Sparsity**
   \[ E[\text{nonzeros } (Q[\tilde{v}, s])] \leq s^2 + \sqrt{d} \]
   \(\rightarrow\) intuitively ensures some compression

3. **Variance bound**
   \[ E[\|Q[v; s]\|_2^2] \leq \left(1 + \frac{\sqrt{d}}{s}\right) \cdot \|v\|_2^2 \]
   \(\rightarrow\) bounded variance \(\rightarrow\) fast convergence
   (Variance increase is 2 for \(s = \sqrt{d}\))
QSGD Compression

**Informal Claim [QSGD]:** There exists a setting of parameters for which QSGD converges at most 2x slower than the full-precision baseline, and sends at most \((32 + 2.8 \cdot d)\) bits per iteration.

QSGD converges similarly to the original, but sends 10x less bits.

Recently [Ramezani et al., 2019] improved on these guarantees.

Proof sketch:

- **Idea1:** Assume we are implementing \(s = \sqrt{d}\) integer quantization levels.
  We notice that very few vector entries can be quantized to the top integers: values are normalized with respect to \(\|v\|_2\), so not all can be large.

- **Idea2:** The resulting “plain text” is a sequence of integers of different frequencies. We can use custom arithmetic coding to encode this sequence efficiently.

Note: The QSGD compression-variance trade-off is tight:
Any algorithm sending < \(B\) bits per round will induce \(d / B\) additional variance.
QSGD can match the \(\Omega\left( d \left( \log d + \log \left( \frac{1}{\varepsilon} \right) \right) \right)\) bit lower bound of [Tsitsiklis & Luo, 1986].
Does it actually work?

- Amazon EC2 p2.xlarge multi-GPU server
- AlexNet model (60M params) × ImageNet dataset × 2 GPUs
- QSGD 4bit quantization (s = 16)
- No additional hyperparameter tuning

Compute

<table>
<thead>
<tr>
<th></th>
<th>SGD</th>
<th>QSGD (d=512)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Communicate</td>
<td>60%</td>
<td>5%</td>
</tr>
<tr>
<td>Compute</td>
<td>40%</td>
<td>95%</td>
</tr>
</tbody>
</table>

AlexNet on ImageNet: Accuracy

Test accuracy (%)
Experiments: “Strong” Scaling

- State-of-the-art image classification on ImageNet, 16-GPU EC2 server
Accuracy vs Time on a Speech Model (LSTM)

- Speech Recognition Dataset (CMU AN4)
- Encoder-Decoder LSTM Model
- **2 GPU nodes**

![Graph showing Accuracy vs Time for a Speech Model (LSTM)]
Summary: Quantization
[1bitSGD, 2014], [QSGD, NeurIPS17], [TernGrad, NeurIPS17], [NUQSGD, 2019]

1. **How much compression?**
   - Usually < 32x, since it’s just bit width reduction
   - Cannot do better without large variance <-> convergence loss

2. **Does it guarantee convergence/accuracy?**
   - **Theory:** Yes (QSGD). Under strong assumptions (1bitSGD).
   - **Practice:** Extensive testing (30’000 node hours) shows QSGD (4bit) preserves accuracy for all neural networks [Grubic et al., EDBT18].

3. **Do they need additional parameter tuning?**
   - TernGrad, 1BitSGD: Yes.
   - QSGD: No.

Method 2: Structured Sparsification
Method 2: Structured Sparsification

[Strom, 2016; Dryden et al., 2017; Aji & Heafield, 2017; Alistarh & Grubic 2018; Lin et al., 2018]

Fix an integer parameter $k$

- Only send top $k$ from each gradient vector, in order of absolute values
- Accumulate the unsent values locally

Transmit $k$ ($32 \log d$) bits

Local Error

Gradient estimator not unbiased!
Method 2: Sparsification
[Dryden et al., 2016; Aji & Heafield, 2017; Lin et al., 2018]

1. How much compression?
   • \( \frac{d}{(k \log d)} \), potentially huge

2. Does it still guarantee convergence?
   • Experimentally: up to 400x compression with no accuracy loss via extremely careful parameter tuning [Lin et al. 2018, ICLR18]
   • Theory: Yes! [Konstantinov et al., NeurIPS 2018]

3. Do they need additional parameter tuning?
   • Yes [Deep Gradient Compression: Lin et al., ICLR18]
   • No [ScaleML: Renggli et al., SuperComputing ’19]
Sparsification with Error Correction Converges

[Konstantinov et al., NeurIPS 2018, journal version in preparation]

**Informal Claim [TopK SGD]:** Under analytic assumptions, given any smooth, (non-convex) function $f$, there exists a learning rate sequence s.t. TopK SGD ensures

$$\min_{t=1,\ldots,T} \mathbb{E}[\|\nabla f(x_t)\|^2]^{T \to \infty} \to 0.$$ 

This suggests that TopK SGD will eventually converge to a local minimum of $f$.

**Convergence rate depends linearly on the “density” parameter $k / d$.**

**Notes:**

1. The above guarantee is the best we can hope for in the non-convex case.
2. The technical argument reveals that TopK is a special case of **asynchronous SGD** [Hogwild!: Niu et al., 2011]
3. **Key for convergence:** how much **gradient norm** is transferred in the **TopK**

In practice, gradients appear to be Gaussian distributed!
Summary so far

Algorithmic methods for scalable distributed machine learning.

- Quantization
- Sparsification

Can provide order-of-magnitude communication reduction!

But how about software support?

Neither method supported by communication libraries (MPI implementations or NVIDIA NCCL)
ScaleML: A Scalable Communication Framework for ML
[Renggli, Ashkboos, Aghagolzadeh, Hoefler, Alistarh; SuperComputing 2019]

• Communication framework with MPI-like semantics
  • Implements distributed AllReduce operations (a.k.a. MPI collectives)
• Native support for quantization and sparsity
• Efficient sparsity support is non-trivial: the underlying sparsity distribution is unknown at runtime
ScaleML: A Scalable Communication Framework for ML
[Renggli, Ashkboos, Aghagolzadeh, Hoefler, Alistarh; SuperComputing19]

- Q1: Can the data become **dense during aggregation**?
  - Depending on this, **we switch to a dense quantized data representation**
- Q2: Is the system **latency-dominated**, or **bandwidth-dominated**?
  - Depending on this, we use **completely different communication patterns**
## Practical Performance

<table>
<thead>
<tr>
<th>System</th>
<th>Model</th>
<th>#Nodes</th>
<th>Sparsity</th>
<th>End-to-end speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSCS Piz Daint</td>
<td>2-Layer LSTM</td>
<td>8</td>
<td>99.5% (+8bit QSGD)</td>
<td>2.6x</td>
</tr>
<tr>
<td>Amazon EC2 Cluster</td>
<td>2-Layer LSTM</td>
<td>8</td>
<td>99.5% (+8bit QSGD)</td>
<td>7.1x</td>
</tr>
</tbody>
</table>

End-to-end training speedup  
(LSTM for Natural Language Understanding / ATIS dataset)

Works well in for a variety of other models/settings.

Is this useful in the real world?
Does it work at scale?

• Microsoft’s Automated Speech Recognition Tool
• State-of-the-art recurrent neural network (RNN)
• Baseline: Fine-Tuned Block-Momentum SGD (BMUF) [Zhu et al., 2016]
• We use 99.5% induced sparsity (k = 0.5%) and 8-bit quantization

We can leverage sparsity and compression in real-world settings as well. ScaleML is compatible with TensorFlow, Pytorch, and Microsoft CNTK.
Summary

Algorithmic methods for scalable distributed machine learning

Quantization  Sparsification  Efficient Aggregation

Trade-offs: compression vs speed vs parametrization.

Distributed machine learning is wide open
Topics I Couldn’t Cover Today

**Asynchronous Distributed Machine Learning**

**Fault-Tolerant Distributed Machine Learning**

**Distributed Learning on Graph Topologies**

**Distributed Sampling & Inference on Graphical Models**
Open Questions

Q1. What is the interaction between convergence and communication topology for optimization algorithms (e.g. SGD)?

Q2. Is there a general set of consistency conditions for distributed machine learning algorithms?

Q3. Can we model distributed learning in social groups?