Distributed Machine Learning: A Brief Overview

Dan Alistarh
IST Austria
Background

The Machine Learning “Cambrian Explosion”

Key Factors:

1. **Large Datasets:**
   - *Millions* of labelled images, *thousands of hours* of speech

2. **Improved Models and Algorithms:**
   - Deep Neural Networks: *hundreds* of layers, *millions* of parameters

3. **Efficient Computation for Machine Learning:**
   - Computational power for ML increased by ~100x since 2010 (Maxwell line to Volta)
   - Gains *almost stagnant* in latest generations (GPU: <1.8x, CPU: <1.3x)
   - Computation times are extremely large anyway (days to weeks to months)

Go-to Solution: **Distribute** Machine Learning Applications to Multiple Processors and Nodes
The Problem

CSCS: Europe’s Top Supercomputer (World 3rd)
- 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 Problem

CSCS: Europe’s Top Supercomputer (World 3rd)
- 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 Problem

CSCS: Europe’s Top Supercomputer (World 3rd)
• 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.
Part 1: Basics
Machine Learning in 1 Slide

\[
\text{argmin}_x f(x) \\
\text{where } f(x) = \sum_{i=1}^{M} \text{loss}(x, e_i) \\
\text{model } x
\]

Solved via optimization procedure, e.g. stochastic gradient descent (SGD).

Notion of “quality,” e.g. squared distance

E.g., neural network or linear model

E.g., classification
Distributed Machine Learning in 1 Slide

$$\text{argmin}_x f(x) = f_1(x) + f_2(x)$$

$$f_1(x) = \sum_{i=1}^{M/2} l(x, ei)$$

$$f_2(x) = \sum_{i=M/2+1}^{M} l(x, ei)$$

This is the (somewhat standard) data parallel paradigm, but there are also model parallel or hybrid approaches.
The Optimization Procedure: Stochastic Gradient Descent

- Gradient descent (GD): \( x_{t+1} = x_t - \eta_t \nabla f(x_t) \).

- Stochastic gradient descent:
  Let \( \bar{g}(x_t) \) = gradient at *randomly chosen* point.
  \( x_{t+1} = x_t - \eta_t \bar{g}(x_t) \), where \( E[\bar{g}(x_t)] = \nabla f(x_t) \).

- Let \( E[||\bar{g}(x) - \nabla f(x)||^2] \leq \sigma^2 \) (variance bound)

**Theorem** [classic]: Given \( f \) convex and \( \ell \)-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 Compromise

• **Mini-batch SGD:**
  Let $\tilde{g}_B(x_t) = \text{stochastic gradient with respect to a set of } B \text{ randomly chosen points.}$

  $$x_{t+1} = x_t - \eta_t \tilde{g}_B(x_t), \text{ where } E[\tilde{g}_B(x_t)] = \nabla f(x_t).$$

• Why is this better?
  • The variance $\sigma^2$ of $\tilde{g}_B(x_t)$ is reduced linearly by $B$ with respect to $\tilde{g}(x_t)$
  • By the previous Theorem, the algorithm will converge in $B$ times less iterations (in the convex case)

**Note:** Convergence is less well understood for non-convex optimization objectives (e.g., neural nets). In this case, it’s known that SGD converges to a local optimum (point where gradient = 0).
SGD Parallelization

\[ \tilde{g}_n = \frac{\sum_i g^i}{n} \]

Aggregation can be performed via:
- Master node ("parameter server")
- MPI All-Reduce ("decentralized")
- Shared-Memory

Stochastic gradient with \( n \) times lower variance

Theory: by distributing, we can perform \( P \) times more work per "clock step." Hence, we should converge \( P \) times faster in terms of wall-clock time.

Embarrassingly parallel?
The Practice

Training very large models efficiently

• Vision
  • ImageNet: 1.3 million images
  • ResNet-152 [He+15]: 152 layers, 60 million parameters
  • Model/update size: approx. 250MB

• Speech
  • NIST2000 Switchboard dataset: 2000 hours
  • LACEA [Yu+16]: 22 LSTM (recurrent) layers, 65 million parameters (w/o language model)
  • Model/update size: approx. 300MB

Data parallel SGD

Compute gradient | Exchange gradient | Update params

Minibatch 1 | Minibatch 2 | Minibatch 3
Data parallel SGD (bigger models)

- Compute gradient
- Exchange gradient
- Update params

Minibatch 1
Minibatch 2
Data parallel SGD (**biggerer** model)

- **Minibatch 1**
  - Compute gradient
  - Exchange gradient
  - Update params

- **Minibatch 2**
  - Compute gradient
  - Exchange gradient
  - Update params
More Precisely: Two major costs

Computing gradient

Exchange gradient

Update params

Communication

Synchronization

Minibatch 1

Minibatch 2
Part 2: Communication-Reduction Techniques
Data parallel SGD (*biggerer* model)
Idea [Seide et al., 2014]: *compress* the gradients...
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 error locally, and apply to next gradient!

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”
Why this shouldn’t work

Let $Q(x)$ be the gradient quantization function.

- Iteration:
  
  \[ x_{t+1} = x_t - \eta_t Q(\nabla f(x_t)) \text{ where } E[\tilde{g}(x_t)] = \nabla f(x_t). \]

- Let:
  
  - $E[||\tilde{g}(x) - \nabla f(x)||^2] \leq \sigma^2$ (variance bound)

\[ E[\tilde{g}(x_t)] \neq \nabla f(x_t) \]

\[ x_{t+1} = x_t - \eta_t Q(\nabla f(x_t)) \]

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

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

\[ E \left[ f \left( \frac{1}{T} \sum_{t=0}^{T} x_t \right) \right] - f(x^*) \leq \varepsilon. \]
Take One: Stochastic Quantization

• Quantization function

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

where \( \xi_i(v_i) = 1 \) with probability \( |v_i|/\|v\|_2 \) and 0 otherwise.

Properties:
1. Unbiasedness:
   \[ E[Q[v_i]] = \|v\|_2 \cdot \text{sgn}(v_i) \cdot |v_i|/\|v\|_2 = \text{sgn}(v_i) \cdot |v_i| \]

2. Second moment (variance) bound:
   \[ E[\|Q[v]\|^2] \leq \|v\|_2 \|v\|_1 \leq \sqrt{n} \|v\|^2 \]

3. Sparsity: If \( v \) has dimension \( n \), then
   \[ E[\text{non-zeroes in } Q(v)] = E[\sum_i \xi_i(v)] \leq \|v\|_1/\|v\|_2 \leq \sqrt{n} \]

Convergence:
\[ E[Q[\tilde{g}(x_i)]] = E[\tilde{g}(x_i)] = \nabla f(x_i) \]

Runtime \( \leq \sqrt{n} \) more iterations

\[ ||v||_2 = 0.447 \]
Compression

• Quantization function

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

where \( \xi_i(v_i) = 1 \) with probability \( |v_i|/\|v\|_2 \) and 0 otherwise.

Original: 32n bits

Compression \( \approx \sqrt{n}/\log n \).

Compressed: \( 32 + \sqrt{n} \log n \) bits

Moral: We’re not too happy:

the \( \sqrt{n} \) increase in number of iterations offsets the \( \sqrt{n}/\log n \) compression.
Take Two: QSGD

[Alistarh, Grubic, Li, Tomioka, Vojnovic, NIPS17]

- Quantization function

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

where

\[ s = 1 \text{ reduces to the two-bit quantization function.} \]

- Note: \( s=1 \) reduces to the two-bit quantization function.
QSGD Properties

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

- Properties
  1. Unbiasedness
     \[ E[Q[v_i; s]] = v_i \]
  2. Sparsity
     \[ E[\|Q(v, s)\|_0] \leq s^2 + \sqrt{n} \]
  3. Second moment bound
     \[ E[\|Q[v; s]\|_2^2] \leq \left( 1 + \min \left( \frac{n}{s^2}, \frac{\sqrt{n}}{s} \right) \right) \cdot \|v\|_2^2 \]  
     (Multiplier only 2 for \( s = \sqrt{n} \))
Two Regimes

**Theorem 1 (constant s):** The expected bit length of the quantized gradient is

\[ 32 + (s^2 + \sqrt{n}) \log n. \]

**Theorem 2 (large s):** For \( s = \sqrt{n} \), the expected bit length of the quantized gradient is

\[ 32 + 2.8 \cdot n, \] and the added variance is **constant**.

- **Idea1:** there can be few large integer values encoded
- **Idea2:** Use **Elias recursive coding** to code integers efficiently

**Theorem [Tsitsiklis&Luo, ‘86]:** Given dimension \( n \), the necessary number of bits for approximating the minimum within \( \varepsilon \) is

\[ \Omega \left( n \left( \log n + \log \left( \frac{1}{\varepsilon} \right) \right) \right). \]

**Matches Thm 2.**

**Original:** 32n bits.
Does it actually work?

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

Compute

\begin{tabular}{ll}
SGD & 60% \\
QSGD 4bit (d=512) & 5% \\
\end{tabular}

Communicate

\begin{tabular}{ll}
SGD & 40% \\
QSGD 4bit (d=512) & 95% \\
\end{tabular}

SGD vs QSGD on AlexNet.
Experiments: “Strong” Scaling

### AlexNet
- 2 GPUs: 1.2 hours
- 4 GPUs: 0.8 hours
- 8 GPUs: 0.6 hours
- 16 GPUs: 0.4 hours
- QSGD 4bit (d=512): 0.2 hours

Time per epoch: 2.5x

### VGG19
- 2 GPUs: 20 hours
- 4 GPUs: 15 hours
- 8 GPUs: 10 hours
- 16 GPUs: 5 hours
- QSGD 4bit (d=512): 2.5 hours

Time per epoch: 3.5x

### ResNet152
- 2 GPUs: 16 hours
- 4 GPUs: 14 hours
- 8 GPUs: 12 hours
- 16 GPUs: 10 hours
- QSGD 4bit (d=512): 8 hours

Time per epoch: 1.8x

### BN-Inception
- 2 GPUs: 2.5 hours
- 4 GPUs: 2.0 hours
- 8 GPUs: 1.5 hours
- 16 GPUs: 1.0 hours
- QSGD 4bit (d=512): 0.5 hours

Time per epoch: 1.3x
Experiments: Accuracy

Across all networks we tried, 4 bits are sufficient. (QSGD report contains full numbers and comparisons.)
Other Communication-Efficient Approaches

Quantization-based methods yield stable, but limited gains in practice
• Usually $< 32x$ compression, since it’s just bit width reduction
• Can’t do much better without large variance [QSGD, NIPS17]

The “Engineering” approach [NVIDIA NCCL]
• Increase network bandwidth, decrease network latency
• New interconnects (NVIDIA, CRAY), better protocols (NVIDIA)

The ”Sparsification” approach [Dryden et al., 2016; Aji et al., 2018]
• Send the “important” components of each gradient, sorted by magnitude
• Empirically gives much higher compression (up to 800x [Han et al., ICLR 2018])

“Large-Batch” approaches [Goyal et al., 2017; You et al., 2018]
• Run more computation locally before communicating (large “batches”)
• Need extremely careful parameter tuning in order to work without accuracy loss
Roadmap

• Introduction
• Basics
  • Distributed Optimization and SGD
• Communication-Reduction
  • Stochastic Quantization and QSGD
• Asynchronous Training
  • Asynchronous SGD
• Recent Work
Two major costs

- Compute gradient
- Exchange gradient
- Update params

Communication

Synchronization
SGD Parallelization

\[ \sum \tilde{g}_n \]

Dataset Partition 1
Dataset Partition 2
Dataset Partition 3
...
Dataset Partition n

Aggregation in Shared Memory
Lock-based? Lock-free?
SGD in Asynchronous Shared Memory

$P$ threads, adversarial scheduler

- Model updated using atomic operations (read, CAS/Fetch-and-add)

Does SGD still converge under asynchronous (inconsistent) iterations?
Define $\tau =$ maximum number of previous updates a scan may miss. Note that $\tau \leq$ maximum interval contention for an operation.
Convergence Intuition

Legend:
- **Blue** = original minibatch SGD
- **Red** dotted = delayed updates

**Adversary’s power:**
- Delay a subset of gradient updates
- Move the delayed updates to delay convergence to the **optimum**

**Adversary’s limitation:**
- $\tau$ is the **maximum delay** between when the step is generated and when it has to be applied

**Theorem [Recht et al., ‘11]:** Under analytic assumptions, asynchronous SGD still converges, but at a rate that is $O(\tau)$ times slower than serial SGD.
Convergence of Asynchronous SGD ("Hogwild")

**Theorem [Recht et al., ‘11]:** Under analytic assumptions, asynchronous SGD still converges, but at a rate that is \( O(\tau) \) times slower than serial SGD.

Lots of follow-up work, tightening assumptions.

The linear dependency on \( \tau \) is tight in general, but can be reduced to \( \sqrt{P\tau} \) by simple modifications [PODC18].

This is a **worst-case bound**: in practice, asynchronous SGD sometimes converges at **the same rate** as the serial version.

Theoretical gains come from the fact that the \( \tau \) slowdown due to async is compensated by the speedup of \( P \) due to parallelism.

More details in Nikola’s talk on Wednesday morning!

Recht et al. “HOGWILD!: A Lock-Free Approach to Parallelizing Stochastic Gradient Descent”, NIPS 2011
Asynchronous Approaches

The Convex Case:
• By now, lock-free is the standard implementation of SGD in shared memory
• Exploit the fact that many large datasets are sparse, so conflicts are rare
• NUMA case is much less well understood

The Non-Convex Case:
• Requires careful hyperparameter tuning to work, and is less popular
• Convergence of SGD in the non-convex case is less well understood, and very little is known analytically [Lian et al, NIPS 2015]

Summary

Most medium-to-large-scale machine learning is distributed.

Communication-efficient and asynchronous learning techniques are fairly common, and are starting to have a sound theoretical basis.

Lots of exciting new questions!
A Sample of Open Questions

What are the notions of *consistency* required by distributed machine learning algorithms in order to converge?

At first sight, *much weaker* than standard notions.

$$\arg\min_x f(x) = f_1(x) + f_2(x)$$

model $x_1 = x + \text{noise}$  
model $x_1 = x + \text{noise}$
A Sample of Open Questions

Can distributed Machine Learning algorithms be Byzantine-resilient?

Early work by [Su, Vaidya], [Blanchard, El Mhamdi, Guerraoui, Steiner]

Non-trivial ideas from both ML and distributed computing sides.

$$\text{argmin}_x f(x) = f_1(x_1) + f_2(x_2)$$
Can distributed Machine Learning algorithms be **completely decentralized**? Early work by e.g. [Lian et al., NIPS 2017], for SGD.