Distributed Machine Learning

Georgios Damaskinos
2018
Machine Learning?
Machine Learning “in a nutshell”
Machine Learning algorithm

Cost Functions

Root Mean Squared Error (RMSE)

\[ \sqrt{\frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2} \]

Root Mean Squared Log Error (RMSLE)

\[ \sqrt{\frac{1}{n} \sum_{i=1}^{n} (\log(p_i + 1) - \log(a_i + 1))^2} \]
Machine Learning algorithm

\[ h_\theta(x) = \sum_{j=0}^{n} \theta_j x_j \]

\[ J_{\text{train}}(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 \]

Repeat {
    \[ \theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)} \]
    (for every \( j = 0, \ldots, n \))
}


Safety?
Safety?

Cost function

Convergence
Machine Learning?
Think big!

“It’s not who has the best algorithm that wins. It’s who has the most data.”

Andrew Ng
Think big!

Example: Image Classification

**Data:**
ImageNet: 1.3 Million training images (224 x 224 x 3)

**Model:**
ResNet-152: 60.2 Million parameters (model size)

Training time (single node):
TensorFlow: **19 days!!**
Think big!

Example: Image Classification

**Data:**
- ImageNet: 1.3 Million training images (224 x 224 x 3)

**Model:**
- ResNet-152: 60.2 Million parameters (model size)

Training time (single node):
  - TensorFlow: **19 days!!**
Distributed Computing
Performance?

Training time (single node):
  TensorFlow: **19 days**

Training time (distributed):
  1024 Nodes (theoretical): **25 minutes**
  CSCS (3rd top supercomputer, 4500+ GPUs, state-of-the-art interconnect:

![Graph showing time to train model with varying number of GPU nodes.](Image)
Performance?

State-of-the-art (ResNet-50): **1 hour** [GP+17]

- Batch size = 8192
- 256 GPUs

Distributed ... how?

Model Parallelism

Data Parallelism
Data Parallelism

Parameter Server: $\frac{1}{4} \sum_{j=1}^{4} W_{i+1,j}$

\[ W_{i+1,1}, W_{i+1,2}, W_{i+1,3}, W_{i+1,4} \]

Machine 1, Machine 2, Machine 3, Machine 4
Batch Learning

\[ \theta \]

\[ \mathcal{P}_1 \]

0

Time

Parameters: \( \theta_t \)

Processors: \( \mathcal{P}_i \)

Dataset: \( \mathcal{D} \)
Batch Learning

\[ \theta \rightarrow \theta_0 \]

\[ P_1 \quad g = \text{calc}(D, \theta_0) \]

\[ g = \text{calc}(D, \theta) : \]
Calculate gradient \( g \) on data \( D \) using parameters \( \theta \)

Parameters: \( \theta_t \)
Processors: \( P_i \)
Dataset: \( D \)
Batch Learning

\[
\begin{array}{c|c|c}
\theta & \theta_0 & \theta_1 \\
\hline
\mathcal{P}_1 & \text{g = calc(\mathcal{D}, \theta_0)} & \theta_1 = \text{up(\theta_0, g)} \\
\end{array}
\]

- **g = calc(\mathcal{D}, \theta)**: Calculate gradient \( g \) on data \( \mathcal{D} \) using parameters \( \theta \)
- **\theta_1 = \text{up(\theta_0, g)}**: Update \( \theta_0 \) using gradient \( g \) to obtain \( \theta_1 \)

**Parameters:** \( \theta_t \)

**Processors:** \( \mathcal{P}_i \)

**Dataset:** \( \mathcal{D} \)
# Batch Learning

<table>
<thead>
<tr>
<th>$\theta$</th>
<th>$\theta_0$</th>
<th>$\theta_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathcal{P}_1$</td>
<td>$g = \text{calc}(\mathcal{D}, \theta_0)$</td>
<td>$\theta_1 = \text{up}(\theta_0, g)$, $g = \text{calc}(\mathcal{D}, \theta_1)$</td>
</tr>
</tbody>
</table>

- **$g = \text{calc}(\mathcal{D}, \theta)$**: Calculate gradient $g$ on data $\mathcal{D}$ using parameters $\theta$.
- **$\theta_1 = \text{up}(\theta_0, g)$**: Update $\theta_0$ using gradient $g$ to obtain $\theta_1$.

**Parameters**: $\theta_t$

**Processors**: $\mathcal{P}_i$

**Dataset**: $\mathcal{D}$
Parallel Batch Learning

- Partition Data
- Parallel Compute on Partitions

\[
\begin{array}{|c|c|}
\hline
\theta & \theta_0 \\
\hline
\mathcal{P}_1 & g_1 = \text{calc}(\mathcal{D}_1, \theta_0) \\
\mathcal{P}_2 & g_2 = \text{calc}(\mathcal{D}_2, \theta_0) \\
\mathcal{P}_3 & g_3 = \text{calc}(\mathcal{D}_3, \theta_0) \\
\hline
\end{array}
\]

Parameters: $\theta_t$
Processors: $\mathcal{P}_i$
Dataset: $\mathcal{D} = \mathcal{D}_1 \cup \mathcal{D}_2 \cup \mathcal{D}_3$
Gradient: $g = g_1 + g_2 + g_3$
Parallel Batch Learning

| \(P_1\) | \(g_1 = \text{calc}(D_1, \theta_0)\) | \(\theta_1 = \text{up}(\theta_0, g)\) |
| \(P_2\) | \(g_2 = \text{calc}(D_2, \theta_0)\) |
| \(P_3\) | \(g_3 = \text{calc}(D_3, \theta_0)\) |

Parameters: \(\theta_t\)
Processors: \(P_i\)
Dataset: \(D = D_1 \cup D_2 \cup D_3\)
Gradient: \(g = g_1 + g_2 + g_3\)
## Parallel Batch Learning

<table>
<thead>
<tr>
<th>$\theta$</th>
<th>$\theta_0$</th>
<th>$\theta_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathcal{P}_1$</td>
<td>$g_1 = \text{calc}(\mathcal{D}_1, \theta_0)$</td>
<td>$\theta_1 = \text{up}(\theta_0, g)$</td>
</tr>
<tr>
<td>$\mathcal{P}_2$</td>
<td>$g_2 = \text{calc}(\mathcal{D}_2, \theta_0)$</td>
<td></td>
</tr>
<tr>
<td>$\mathcal{P}_3$</td>
<td>$g_3 = \text{calc}(\mathcal{D}_3, \theta_0)$</td>
<td></td>
</tr>
</tbody>
</table>

**Parameters:** $\theta_t$

**Processors:** $\mathcal{P}_i$

**Dataset:** $\mathcal{D} = \mathcal{D}_1 \cup \mathcal{D}_2 \cup \mathcal{D}_3$

**Gradient:** $g = g_1 + g_2 + g_3$
Parallel **Synchronous** Mini-Batch Learning

- More frequent updates

Parameters: $\theta_t$

Processors: $P_i$

Mini-batches: $B_t = B_1^t \cup B_2^t \cup B_3^t$

Gradient: $g = g_1 + g_2 + g_3$
## Parallel Asynchronous Mini-Batch Learning

<table>
<thead>
<tr>
<th>( \theta )</th>
<th>( \theta_0 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( P_1 )</td>
<td></td>
</tr>
<tr>
<td>( P_2 )</td>
<td></td>
</tr>
<tr>
<td>( P_3 )</td>
<td></td>
</tr>
</tbody>
</table>

Parameters: \( \theta_t \)
Processors: \( P_i \)
Mini-batches: \( B_j \)
Gradient: \( g_k \)
Parallel Asynchronous Mini-Batch Learning

<table>
<thead>
<tr>
<th>$\theta$</th>
<th>$\theta_0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathcal{P}_1$</td>
<td>$g_1 = c(B_1, \theta_0)$</td>
</tr>
<tr>
<td>$\mathcal{P}_2$</td>
<td>$g_2 = c(B_2, \theta_0)$</td>
</tr>
<tr>
<td>$\mathcal{P}_3$</td>
<td>$g_3 = c(B_3, \theta_0)$</td>
</tr>
</tbody>
</table>

Time

Parameters: $\theta_t$
Processors: $\mathcal{P}_i$
Mini-batches: $B_j$
Gradient: $g_k$
Parallel Asynchronous Mini-Batch Learning

<table>
<thead>
<tr>
<th>θ</th>
<th>θ₀</th>
<th>θ₁</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathcal{P}_1$</td>
<td>$g_1 = c(B_1, \theta_0)$</td>
<td>$\theta_1 = u(\theta_0, g_1)$</td>
</tr>
<tr>
<td>$\mathcal{P}_2$</td>
<td>$g_2 = c(B_2, \theta_0)$</td>
<td></td>
</tr>
<tr>
<td>$\mathcal{P}_3$</td>
<td>$g_3 = c(B_3, \theta_0)$</td>
<td></td>
</tr>
</tbody>
</table>

Parameters: $\theta_t$
Processors: $\mathcal{P}_i$
Mini-batches: $B_j$
Gradient: $g_k$
Parallel Asynchronous Mini-Batch Learning

<table>
<thead>
<tr>
<th>θ</th>
<th>θ₀</th>
<th>θ₁</th>
</tr>
</thead>
<tbody>
<tr>
<td>( P_1 )</td>
<td>( g_1 = c(B_1, \theta_0) )</td>
<td>( \theta_1 = u(\theta_0, g_1) )</td>
</tr>
<tr>
<td>( P_2 )</td>
<td>( g_2 = c(B_2, \theta_0) )</td>
<td></td>
</tr>
<tr>
<td>( P_3 )</td>
<td>( g_3 = c(B_3, \theta_0) )</td>
<td></td>
</tr>
</tbody>
</table>

Parameters: \( \theta_t \)
Processors: \( P_i \)
Mini-batches: \( B_j \)
Gradient: \( g_k \)
Parallel Asynchronous Mini-Batch Learning

<table>
<thead>
<tr>
<th>$\theta$</th>
<th>$\theta_0$</th>
<th>$\theta_1$</th>
<th>$\theta_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathcal{P}_1$</td>
<td>$g_1 = c(B_1, \theta_0)$</td>
<td>$\theta_1 = u(\theta_0, g_1)$</td>
<td>$g_1 = c(B_4, \theta_1)$</td>
</tr>
<tr>
<td>$\mathcal{P}_2$</td>
<td>$g_2 = c(B_2, \theta_0)$</td>
<td>$\theta_2 = u(\theta_1, g_2)$</td>
<td></td>
</tr>
<tr>
<td>$\mathcal{P}_3$</td>
<td>$g_3 = c(B_3, \theta_0)$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Parameters: $\theta_t$  
Processors: $\mathcal{P}_i$  
Mini-batches: $B_j$  
Gradient: $g_k$
### Parallel Asynchronous Mini-Batch Learning

<table>
<thead>
<tr>
<th>$\theta$</th>
<th>$\theta_0$</th>
<th>$\theta_1$</th>
<th>$\theta_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathcal{P}_1$</td>
<td>$g_1 = c(B_1, \theta_0)$</td>
<td>$\theta_1 = u(\theta_0, g_1)$</td>
<td>$g_1 = c(B_4, \theta_1)$</td>
</tr>
<tr>
<td>$\mathcal{P}_2$</td>
<td>$g_2 = c(B_2, \theta_0)$</td>
<td></td>
<td>$\theta_2 = u(\theta_1, g_2)$</td>
</tr>
<tr>
<td>$\mathcal{P}_3$</td>
<td>$g_3 = c(B_3, \theta_0)$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Parameters:** $\theta_t$  
**Processors:** $\mathcal{P}_i$  
**Mini-batches:** $B_j$  
**Gradient:** $g_k$
Parallel Asynchronous Mini-Batch Learning

<table>
<thead>
<tr>
<th>( \theta )</th>
<th>( \theta_0 )</th>
<th>( \theta_1 )</th>
<th>( \theta_2 )</th>
<th>( \theta_3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( P_1 )</td>
<td>( g_1 = c(B_1, \theta_0) )</td>
<td>( \theta_1 = u(\theta_0, g_1) )</td>
<td>( g_1 = c(B_4, \theta_1) )</td>
<td></td>
</tr>
<tr>
<td>( P_2 )</td>
<td>( g_2 = c(B_2, \theta_0) )</td>
<td>( \theta_2 = u(\theta_1, g_2) )</td>
<td>( g_2 = c(B_5, \theta_2) )</td>
<td></td>
</tr>
<tr>
<td>( P_3 )</td>
<td>( g_3 = c(B_3, \theta_0) )</td>
<td></td>
<td>( \theta_3 = u(\theta_2, g_3) )</td>
<td></td>
</tr>
</tbody>
</table>

Parameters: \( \theta_t \)
Processors: \( P_i \)
Mini-batches: \( B_j \)
Gradient: \( g_k \)
Parallel Asynchronous Mini-Batch Learning

- Gradients computed using **stale** parameters
- Increased utilization
- Central lock
Distributed ML

- Parallelism
  - Model
  - Data
- Learning
  - Synchronous
  - Asynchronous
Distributed ML: Challenges

1. Scalability
2. Privacy
3. Security
Scalability - Asynchrony

After completing a mini-batch, 25% chance of delaying

Delay (in seconds) sampled from \( \max(\mathcal{N}(\mu, (\mu/5)^2), 0) \)

Avg. time per mini-batch = 0.62 s
Scalability - Communication

ImageNet classification (ResNet-152):
Model/update size = \(~ 250\text{MB}\)
Scalability - Communication
Scalability - Communication
Scalability - Communication
Scalability - Communication
Scalability - Communication

ImageNet classification (ResNet-152):
Mode/update size = ~ 250MB

Compression
- Distillation [PPA+18]
- Quantization [DGL+17]
  - SignSGD [BJ+18]

Distributed ML: Challenges

1. Scalability
   a. Asynchrony
   b. Communication efficiency

2. Privacy

3. Security
• Medical data

• Photos

• Search logs
Privacy

Differential Privacy
● Decentralized Learning [BGT+18]
● Compression <-> DP [AST+18]

Local Privacy
● MPC

Distributed ML: Challenges

1. Scalability
   a. Asynchrony
   b. Communication efficiency

2. Privacy
   a. Differential Privacy
   b. Local Privacy

3. Security
Security: Byzantine worker

\[ x' = x - \eta \nabla x \]

Examples:
- crash
- software bug
- corrupted data
- security flaw
Security: Synchronous BFT

<table>
<thead>
<tr>
<th></th>
<th>$\theta_0$</th>
<th>$\theta_1$</th>
<th>$\theta_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>$g_1 = c(B_1^1, \theta_0)$</td>
<td>$\theta_1 = u(\theta_0, g)$</td>
<td>$g_1 = c(B_2^1, \theta_1)$</td>
</tr>
<tr>
<td>$P_2$</td>
<td>$g_2 = c(B_2^2, \theta_0)$</td>
<td>$g_2 = c(B_2^2, \theta_1)$</td>
<td>$g_2 = c(B_3^2, \theta_0)$</td>
</tr>
<tr>
<td>$P_3$</td>
<td>$g_3 = c(B_3^3, \theta_0)$</td>
<td>$g_3 = c(B_3^3, \theta_1)$</td>
<td>$g_3 = c(B_3^3, \theta_0)$</td>
</tr>
</tbody>
</table>

Parameters: $\theta_t$

Processors: $P_i$

Mini-batches: $B_t = B_t^1 \cup B_t^2 \cup B_t^3$

Gradient: $g = g_1 + g_2 + g_3$
Security: Synchronous BFT

Krum

- Byzantine resilience against f/n workers, $2f + 2 < n$
- Provable convergence (i.e., safety)

How?
1. Worker i: score(i) = \[
\sum_{n-f-2 \text{ closest vectors to } G_i} ||G_i - G_j||^2
\]
   Select gradient with minimum score
2. m-Krum

Majority + Squared distance-based decision -> BFT
Security: Asynchronous BFT

<table>
<thead>
<tr>
<th>$\theta$</th>
<th>$\theta_0$</th>
<th>$\theta_1$</th>
<th>$\theta_2$</th>
<th>$\theta_3$</th>
<th>$\theta_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>$g_1 = c(B_1, \theta_0)$</td>
<td>$\theta_1 = u(\theta_0, g_1)$</td>
<td>$g_1 = c(B_4, \theta_1)$</td>
<td>$\theta_4 = u(\theta_3, g_1)$</td>
<td>$g_1 = c(\theta_4)$</td>
</tr>
<tr>
<td>$P_2$</td>
<td>$g_2 = c(B_2, \theta_0)$</td>
<td>$\theta_2 = u(\theta_1, g_2)$</td>
<td>$g_2 = c(B_5, \theta_2)$</td>
<td>$\theta_5 = u(\theta_4)$</td>
<td>$g_2 = c(\theta_5)$</td>
</tr>
<tr>
<td>$P_3$</td>
<td>$g_3 = c(B_3, \theta_0)$</td>
<td>$\theta_3 = u(\theta_2, g_3)$</td>
<td>$g_3 = c(B_6, \theta_3)$</td>
<td>$\theta_6 = u(\theta_5)$</td>
<td>$g_3 = c(\theta_6)$</td>
</tr>
</tbody>
</table>

Parameters: $\theta_t$
Processors: $P_i$
Mini-batches: $B_j$
Gradient: $g_k$
Security: Asynchronous BFT

Kardam

- Byzantine resilience against f/n workers, 3f < n
- Optimal slowdown: \( \frac{n-2f}{n-f} \leq SL \leq \frac{n-f}{n} \)
- Provable (almost sure) convergence (i.e., safety)

How?
1. Lipschitz Filtering Component => Byzantine resilience
2. Staleness Dampening Component => Asynchronous convergence

Asynchrony can be viewed as Byzantine behavior
Distributed ML: Challenges

1. Scalability
   a. Asynchrony
   b. Communication efficiency

2. Privacy
   a. Differential Privacy
   b. Local Privacy

3. Security
   a. Synchronous BFT
   b. Asynchronous BFT
Distributed ML: Frameworks

Deep Learning Frameworks 2017

- Google
- Amazon AWS
- Microsoft
- mxnet
- GLUON
- CNTK
- theano
- Caffe
- Caffe2
- PyTorch
- torch

Facebook
Tensorflow: Why?

Popularity

Deep Learning Framework Power Scores 2018

https://towardsdatascience.com/deep-learning-framework-power-scores-2018-23607ddf297a
Tensorflow: Why?

Support

- Visualization tools
- Documentation
- Tutorials
Tensorflow: Why?

Portability - Flexibility - Scalability
Tensorflow: What is it?

- Dataflow graph computation
- Automatic differentiation (also for while loops [Y+18])

Tensor?

Multidimensional array of numbers

Examples:
- A scalar
- A vector
- A matrix
DataFlow?

- Computations are **graphs**
  - Nodes: *Operations*
  - Edges: *Tensors*

- Program phases:
  - Construction: create the graph
  - Execution: push data through the graph
Tensorflow VS DataFlow Frameworks

- Batch Processing
- Relaxed consistency
- Simplicity
  - No join operations
  - Input diff => new batch
Architecture
Learning

(a) Asynchronous replication

(b) Synchronous replication

(c) Synchronous w/ backup worker
TensorFlow BFT ? No!

How can we make it BFT?

[Damaskinos G., El Mhamdi E., Guerraoui R., Guirguis A., Rouault S.]