DISTRIBUTED MACHINE LEARNING ON BIG DATA

Varun Chandola
Machine Learning

- **Learning from data**
  - Distill knowledge from raw data – turn into predictions, actionable hypotheses

- **Many practical applications**
  - Social network analysis
  - Customer behavior prediction
  - Text, image, video interpretation
  - Understanding disease and treatment pathways
  - Self driving cars
  - Cybersecurity
General Practice

• Choose ML algorithm
  • Graphical model, Deep Learning, Sparse Coding, etc.
• Write ML program implementation of the algorithm for an application-specific instance
  • Using a generic high-level programming language
• Example: A neural network for distinguishing between different types of handwritten digits
• Ideally the program will be hardware-agnostic and ML-explicit
  • Will give the same answer on a given data, regardless of hardware choices
Distributed ML

- How to scale ML algorithms to big data?
- Main challenge:
  - Given P computing nodes instead of 1, get P-fold speedup in time
- However, this is typically not observed
Thinking about Distributed ML

• Do I want **faster convergence** or **higher throughput**?

• Historically, scalability research in ML has focused on faster convergence (more on this later)
  • Assume networks are infinitely fast and there are no faults

• Market demands higher throughput (total time taken to run) and fault tolerance
  • Assumes that all algorithms can be correctly implemented under any programming abstraction (MapReduce or Vertex Programming)
What do we do?

- Build an end-to-end ML specific stack
  - Petuum, Parameter Server
- Adapt ML methods to run on general-purpose Big Data software platforms:
  - Hadoop, Spark (*MapReduce*) – Easy to use programming interface, but slower than ML specific systems
    - Check out Mahout and MLI
  - GraphLab, Pregel (*Vertex Programming*) – Faster, but not every algorithm can be conceived as vertex programs
- Use libraries that use low-level utilities for ML such as stochastic proximal descent, coordinate descent, MCMC, etc.
Typical ML Model

$$\max_A \mathcal{L}(\mathbf{x}, A) \text{ OR } \min_A \mathcal{L}(\mathbf{x}, A),$$

where

$$\mathcal{L}(\mathbf{x}, A) = f (\{x_i, y_i\}_{i=1}^N; A) + r(A).$$

- Iterative convergent ML algorithm

$$A(t) = F (A(t - 1), \Delta \mathcal{L}(A(t - 1), \mathbf{x}))$$

- Each ML algorithm has its own $f$, $r$, $\mathbf{x}$, and $A$.
- Deep learning
  - Massive parameter space $A$
  - Complex $f$ (recursive)
- Structured sparse regression models
  - Complex $r$
- The volume of data, $\mathbf{x}$, poses a challenge for all algorithms
How are ML algorithms unique?

• Why can’t we use principles that have been used by HPC and parallel computing community for decades?

• Unique properties
  1. Dynamic structural dependencies
  2. Non-uniform convergence
  3. Error tolerance
  4. Compact updates
How to parallelize?

- **Data** parallelism vs. **Model** parallelism

\[ \bar{\theta}^{t+1} = \bar{\theta}^t + \Delta_f \bar{\theta}(D) \]

**Data Parallel**

\[ D \equiv \{ D_1, D_2, \ldots, D_n \} \]

\[ D_i \perp D_j \mid \theta, \forall i \neq j \]

**Model Parallel**

\[ \bar{\theta} = [\bar{\theta}_1^T, \bar{\theta}_2^T, \ldots, \bar{\theta}_k^T]^T \]

\[ \bar{\theta}_i \not\equiv \bar{\theta}_j \mid D, \exists (i, j) \]
HOW TO DESIGN A GOOD ML SYSTEM FOR BIG DATA?
Distributing the computation

- Scheduling and balancing workload
Bridging computation and communication

Worker 1
Worker 2
Worker 3
Worker 4

One Iteration
Synchronization Barrier
Worker 1
Worker 2
Worker 3
Worker 4

Staleness Threshold $s = 3$
Worker 1 forced to stop
until worker 2 catches up
Managing communication

- Do not have to send updates to every other node
- Can wait before sending updates
- Update prioritization
Conclusions

• Designing a distributed ML system requires a good understanding of the properties of ML algorithms
• Very different from traditional scientific algorithms
• Need to take care of three fundamental principles
• Petuum – based on the three principles (talk on October 3rd)