ECE 901: Large-scale Machine Learning and Optimization **Fall 2016**

Lecture 15 — 10/25

Lecturer: Dimitris Papailiopoulos *Scribe: Charles Kwong*

Note: These lecture notes are still rough, and have only have been mildly proofread.

15.1 Introduction

In this lecture, we will talk about scalable Stochastic Gradient Descent (SGD) algorithms. Many machine learning problems that we have visited so far can be solved in *polynomial time*. However, in a big data analytics setting, where the sample size of the data, i.e. n, can be astronomically large, and the dimension of the data, i.e. d, can be intimidatingly high, polynomial time *solvability* does not necessarily imply *scalability*.

A scalable algorithm hence must be fast. Ideally, the desirable time complexity should be linear with respect to n and d, i.e.

$$t = O(nd).$$

Furthermore, a scalable algorithm must also be easily *parallelizable*, with a decreasing time complexity as the number of cores, p, available increases. i.e. suppose a serial version of the algorithm runs in time T, a parallel version ran by p cores should ideally have time complexity:

$$t = O(T/p).$$

Last but by no mean the least, it goes without saying that the central goal of a scalable algorithm is to deliver an *accurate* solution to the problem of interest. Among accuracy, speed, and parallelizability, machine learning practitioners evaluate the performance of the algorithms measured by these three criteria, as illustrated in Figure 15.1.

15.2 Stochastic Gradient Descent

Stochastic Gradient Descent (SGD) algorithm is an attractive option for solving many of the important machine learning problems, such as matrix completion, topic modeling, spectral analysis, graph clustering, perceptron learning, neural network (back propagation), principal component analysis (Oja's iteration), and least mean square filter. In many learning scenarios, SGD can deliver accurate solutions efficiently. Furthermore, SGD also demonstrates many desirable theoretical and empirical properties such as robustness to noise, simple implementation, small computational footprints, and stability. Let x denote the parameterizaiton

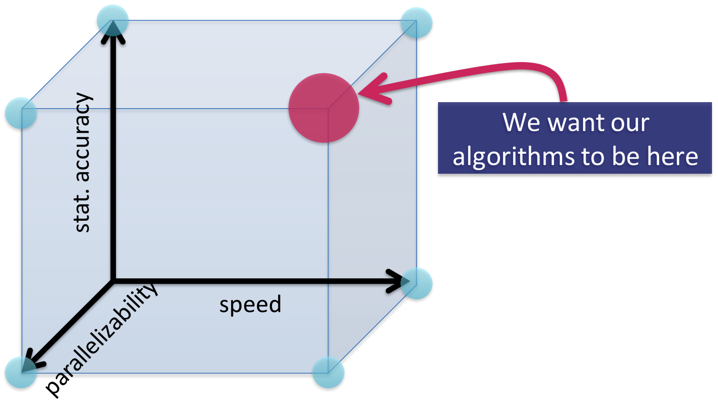

Figure 15.1: Performance Trade-off of a Machine Learning Algorithm

of a learning problem, SGD solves the following optimization problem,

$$\arg\min_{x\in\mathbb{R}^d} \sum_{i=1}^{n} f_i(x),$$

where $f_i(x)$ is the value of a loss function that is only dependent on the i^{th} data point of the dataset. At iteration k, suppose x_k is available, the update rule of SGD is given by:

$$x_{k+1} = x_k - \gamma_k \nabla f_{s_k}(x_k),$$

where $\gamma_k > 0$ is the step length parameter, $\nabla f_{s_k}(x_k)$ is the gradient of $f_{s_k}(x_k)$, and $s_k \in \{1, 2, \cdots, n\}$ is an index randomly drawn at the k^{th} iteration in order to determine the data point on which the loss function and its gradient will be evaluated at the k^{th} iteration.

15.3 Scaling Up SGD

As shown in Figure 15.1, a scalable algorithm must excel in accuracy, speed, and parallelizability. While a classic SGD algorithm can deliver accurate solution efficiently (as described in Section 15.2), it is inherently serial. We therefore have a technical gap to bridge: *can we parallelize inherently serial algorithms?*

To answer this question, in what follows, we will consider an architecture with multiple cores and shared memory. A parallel version of SGD will store the value of the current parameterization in the shared memory. Each core will read, write the shared memory and perform SGD autonomously in order to update the parameterization, as illustrated by Figure 15.2. Without any coordination among cores, two issues might occur. On one hand, the parameterization read by one core from the shared memory might be *obsolete* as other cores might have computed a newer version of the parameteraization. On the other hand, a newer version of the parameterization might be *overwritten* by a core that updates the shared memory with a parameterization that is based on *obsolete* information.

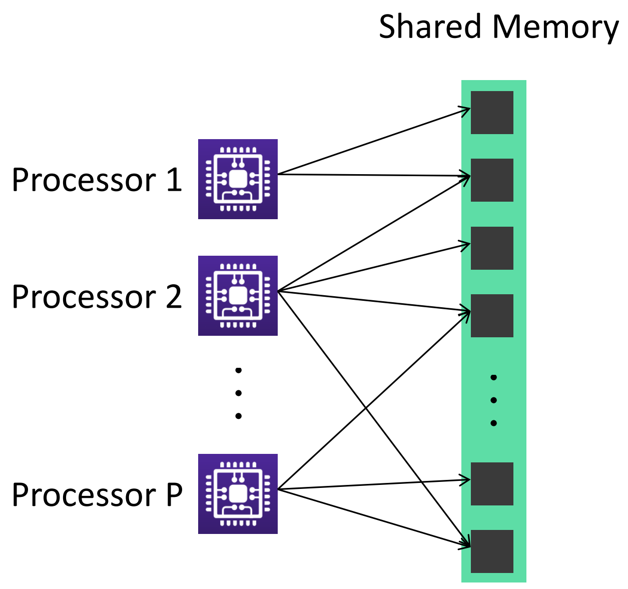

Figure 15.2: Parallel SGD with an architecture with multiple cores and shared memory.

To address these issues, a line of research has been leveraging synchronization, with a focus on coordination among cores and applying locks on the parameterization stored in the shared memory during certain stages of the computation. Many coordination and lock based methods suffer from synchronization and communication overhead. On the contrary, another line of research focuses on the so-called *lock-free* approaches, resulting in the field of asynchronous optimization with literate support dates back to the 1960s [Zinkevich et al., 2010, 2009, Bertsekas and Tsitsiklis, 1989, Tsitsiklis and Bertsekas, 1986, Chazan and Miranker, 1969]. A comparison of these two type of methods is shown in Figure 15.4.

15.4 Lock-Free Parallelization of SGD

In many machine learning problems that can be solved by SGD, the value of $f_i(x)$, given $i \in \{1, 2, \cdots, n\}$, is usually dependent only on a handful of components in x, while the dimension of x can be relatively high. Recht et al. [2011] exploit this kind of *sparsity* and propose an algorithm called HogWild!, which is a lock-free parallelization of SGD with *provable* optimizational guarantee under the aforementioned sparsity assumption.

Let $e_i \subseteq \{1, 2, \cdots, n\}$ be a subset of indices on which the value of $f_i(x)$ depend. i.e. given e_i and a subvector x_{e_i}, which is constructed by the components of x indexed by e_i, the value of $f_i(x)$ is completely determined without knowing the values of other components that are not indexed by $f_i(x)$. We denote $E = \{e_i \mid i \in \{1, 2, \cdots, n\}\}$, and $V = \{x_1, x_2, \cdots, x_d\}$. We can use a *hypergraph*[1], denoted as (V, E), to represent the dependency between the value of $f_i(x)$ and the components of x. Figure 15.3 visualizes a hypergraph. The set of square nodes

[1]A hypergraph is a generalization of a graph in which an edge can join any number of vertices.

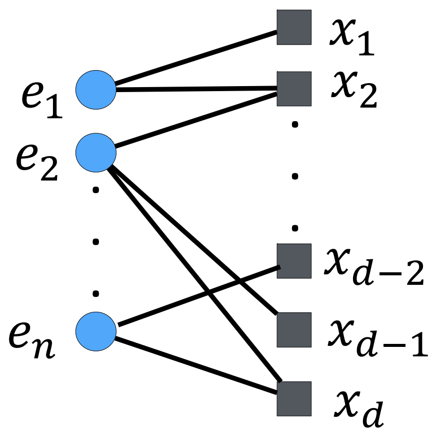

Figure 15.3: A visualization of hypergraph. The set of square nodes represent V. Each blue node represents a grouping of indices that constructs the elements in E. Since the sample size of the dataset is n, the cardinality of E is n, i.e. card $E = n$. Since the dimension of the parameterization is d, we have card $V = d$.

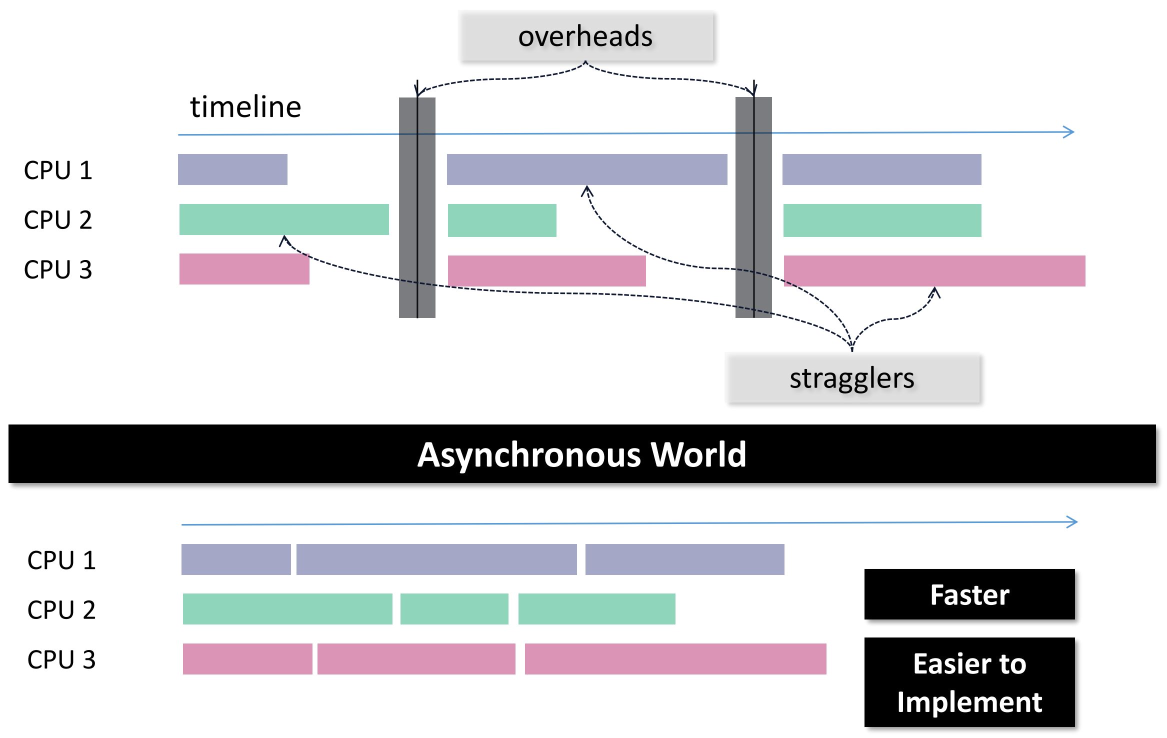

Figure 15.4: A case against synchronization.

Algorithm 1 HogWild!

```
Require: $\mathcal{D}$.
Ensure: $x$.
 1: Initialize $x$ in shared memory.
 2: Compute $E$ from $\mathcal{D}$.
 3: for $i \in \{1, 2, \cdots, p\}$ do in parallel
 4:     while true do
 5:         if Stopping criteria satisfied then
 6:             break.
 7:         end if
 8:         Sample $j \in \{1, 2, \cdots, n\}$.
 9:         Get $x$ from shared memory, choose step length $\gamma$.
10:         Compute $f_j(x; e_j)$, and $\nabla f_j(x; e_j)$.
11:         for $k \in e_j$ do
12:             $x_k \leftarrow x_k - \gamma \nabla_k f_j(x; e_j)$.
13:         end for
14:     end while
15: end for
```

represent V. Each blue node represents a grouping of indices that constructs the elements in E. For example, $e_1 = \{1, 2\}$. Therefore, $f_1(x)$, which is the value of the loss function evaluated on the first data point in the dataset, is dependent only on the first component and the second component of the parameterization. With these notation, we can rewrite the objective function as:

$$\arg\min_{x \in \mathbb{R}^d} \sum_{i=1}^{n} f_i(x; e_i),$$

where $f_i(x; e_i)$ emphasizes the fact that f_i is only dependent on the components of x indexed by e_i. We use $\mathcal{D}$ to denote the dataset. The algorithm of HogWild! is given in Algorithm 1. Note that in Step 10, since $f_i(x; e_i)$ is only dependent on e_i, $\nabla f_i(x; e_i)$ is also only dependent on e_i.

When $e_i \cap e_j = \emptyset$, where $i \neq j$, and $i, j \in \{1, 2, \cdots, n\}$, performing SGD in parallel using $f_i(x)$ and $f_j(x)$ in different cores respectively is equivalent to performing the two iterations in a serial fashion. However, when $e_i \cap e_j \neq \emptyset$, performing SGD in parallel will introduce conflicts. Nonetheless, with the sparsity assumption, one can show that the occurrences of such conflicts are rare and the convergence of HogWild! will not be hindered much by conflicts, resulting in *linear* speedup with the number of processors on many common sparse learning problems.

15.5 Convergence Analysis

Due to parallelization, the theoretical convergence analysis of HogWild! is incompatible with the convergence analysis of SGD. To prove that Parallel SGD and Serial SGD have similar convergence rates for given number of samples, new theoretical general framework are needed for asynchronous lock-free Algorithms. Two measures of performance are of interest in the analysis:

$$\begin{aligned}\text{speedup} &= \frac{\text{Time of serial } \mathcal{A} \text{ to accuracy } \epsilon}{\text{Time of parallel } \mathcal{A} \text{ to accuracy } \epsilon},\\ \text{worst case speedup} &= \frac{\text{bound on number of iterations of SGD to } \epsilon}{\text{bound on number of interations of Parallel SGD to } \epsilon}.\end{aligned}$$

In the next lecture, we will talk about challenges arise in the convergence analysis of asynchronous lock-free Algorithms in details.

References

D. P. Bertsekas and J. N. Tsitsiklis. Convergence rate and termination of asynchronous iterative algorithms. In *Proceedings of the 3rd international conference on Supercomputing*, pages 461–470. ACM, 1989.

D. Chazan and W. Miranker. Chaotic relaxation. *Linear algebra and its applications*, 2(2): 199–222, 1969.

B. Recht, C. Re, S. Wright, and F. Niu. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in Neural Information Processing Systems*, pages 693–701, 2011.

J. Tsitsiklis and D. Bertsekas. Distributed asynchronous optimal routing in data networks. *IEEE Transactions on Automatic Control*, 31(4):325–332, 1986.

M. Zinkevich, J. Langford, and A. J. Smola. Slow learners are fast. In *Advances in Neural Information Processing Systems*, pages 2331–2339, 2009.

M. Zinkevich, M. Weimer, L. Li, and A. J. Smola. Parallelized stochastic gradient descent. In *Advances in neural information processing systems*, pages 2595–2603, 2010.