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

Lecture 16 — 10/27

Lecturer: Dimitris Papailiopoulos *Scribe: Blake Mason & Scott Sievert*

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

16.1 Introduction

Today we'll study an algorithm that runs on a single machine but on multiple cores. We will analyze the case were there are P processors that all read from a shared memory. In recent years we have seen that personal computers tend to follow this exact setup. Often times we would like to perform minimization on a separable function:

$$\hat{x} = \arg\min_x \sum_{e \in \mathcal{E}} f_e(x)$$

Often times we perform stochastic gradient descent (SGD) to solve this minimization as the objective function is separable. This is attractive with the available hardware because recent algorithm [5, 4] can compute these in parallel.

The convergence of running these algorithm asynchronously depends on sparse support of f_e. As mentioned in the last lecture, we define the function-variable graph and draw an edge between nodes f_{e_i} and x_j when x_j is in the support of f_{e_i}. This is has many applications including matrix factorization/composition, graph cuts, topic modeling and more.

HOGWILD! is a method to run the above SGD asynchronously [5]. The algorithm for each processor is shown in Algorithm 1. This has shown significant speedups as shown in Figure 16.1 and is used by Google's Downpour [2] and Microsoft's Project Adam [1].

The main focus of this lecture is analyzing these types of asynchronous algorithms.

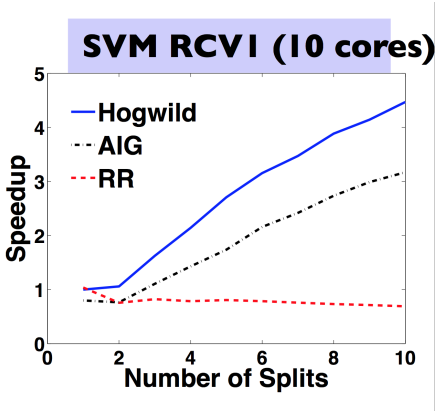

Figure 16.1. The speedup seen by running various algorithms on with different number of splits (analogous to cores)

sample function f_i;
$x =$ read shared memory;
$g = -\gamma \cdot \nabla f_i(x)$;
for *v in the support of f* **do**
  $x_v \leftarrow x_v + g_v$
end

Algorithm 1: The HOGWILD! algorithm for each core

16.2 Main idea

If the algorithm in Figure 1 is being run, we can view the inputs x_e as noisy inputs $\widehat{x}_e$ [3]. When running an algorithm $\mathcal{A}$ on the iterate x where n is some noise, we can view the iterate x as noisy:

$$\mathcal{A}_{\text{async}}(x) = \mathcal{A}_{\text{serial}}(x + n)$$

We also know that writes to memory are both commutative and atomic. That is we know that if we write objects A and B to memory in any order or time, we know that memory will contain $A + B$. We can then represent the iterate at iteration T as

$$x_T = x_0 - \gamma \nabla f_{s_0}(\widehat{x}_0) - \cdots - \gamma \nabla f_{s_{T-1}}(\widehat{x}_{T-1})$$

because the combination of atomic writes and commutativity of addition. This then poses two questions:

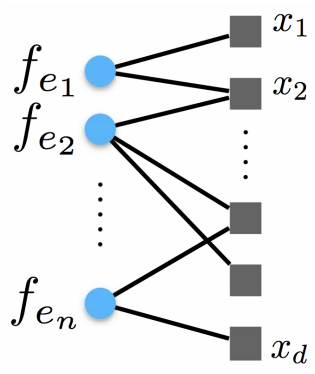

Figure 16.2. The sparsity graph. In this, f_{e_1} depends on variable x_1 and x_2.

1. Where does the noise come from in $\widehat{x}_k$?
2. How strong is this noise?

16.3 Noise origin

We want to analyze noisy stochastic gradient descent,

$$x_{k+1} = x_k - \gamma \cdot \nabla f_{s_k}(\widehat{x}_k)$$

Elementary analysis using m-strong convexity assumptions on f shows that

$$\mathbb{E}\left[\|x_{k+1} - x^\star\|^2\right] \leq (1 - \gamma m) \cdot \mathbb{E}\left[\|x_k - x^\star\|^2\right] + \gamma^2 \mathbb{E}\left[\|\nabla f_{s_k}(\widehat{x}_k\|^2\right]$$

If both terms above are $\mathcal{O}(\gamma^2 M^2)$, noisy SGD gets same rates as SGD up to a multiplicative constance. Let's formulate asynchrony noise mathematically before we bound the size.

16.4 Understanding asynchrony noise

Asynchrony noise comes from the fact that parallel workers can read and write at any time. We will assume the worst case scenario and assume that every shared variable in the support of two iterates has a read/write conflict.

We will assume that no more that τ samples are processed while a core is processing one. If all the cores run at similar speeds this is approximately the number of cores although that could be different for straggler nodes.

Then we know that if the computation for s_i is done before s_k is sampled, its gradient contribution is recorded in shared RAM when another thread starts working on s_k. If s_i and s_k are concurrently processed and there is a conflict in the support, then the gradient contributions of s_i are only partially recorded when a thread starts working on s_k.

For each sample the difference of the noisy iterate $\widehat{x}_k$ and actual iterate x is only caused by other cores that are processing current samples. Because we only process τ samples at once and $\forall i$ the supports of s_i and s_k overlap,

- If s_i is sampled before s_k it overlaps $\iff i \geq k - \tau$
- If s_i is sampled after s_k it overlaps $\iff i \leq k + \tau$

Hence

$$\widehat{x}_k - x_k = \gamma \sum_{i=k-\tau, i\neq k}^{i=k+\tau} S_i^k \nabla f_{s_i}(\widehat{x}_i)$$

where S^k is a diagonal matrix with $S_{ii} = \{\pm 1, 0\}$.

Now the question: how small is asynchrony noise? If $\mathbb{E}\left[\|\nabla f_{s_k}(\widehat{x}_k\|^2\right] = \mathcal{O}(M^2)$ then we can say that this noisy SGD gets the same rates as SGD up to multiplicative constants.

16.5 The Effect of Asychrony Noise

The main goal of this analysis will be to bound the effect of the asynchrony noise implicit in HOGWILD! . If we can show that the error due to asychrony noise is at worst on the same order as the error due to our noisy estimate of the gradient, then we can can expect similar performance guarantees as serial SGD, up to multiplicative constants. Specifically, we wish to bound

$$\gamma \mathbb{E}\{< x_k - \hat{x}_k, \nabla f_{s_k}(\hat{x}_k) >\} = \gamma^2 \mathbb{E}\Big\langle \sum_{i=k-\tau, i\neq k}^{k+\tau} S_i^k \nabla f_{s_i}(\hat{x}_i), \nabla f_{s_k}(\hat{x}_k) \Big\rangle$$

.

We need for this entire term to be less than $\gamma^2 M^2$. As a general idea, concurrently processed sample only interfere if they affect the same variables.

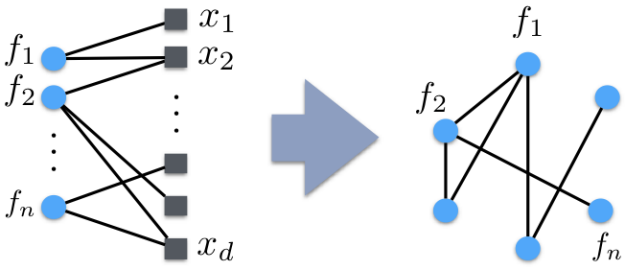

Figure 16.3. The function-variable graph and sparsity conflict graph

In the above image, samples 1 and 2 interfere as they both talk to index x_2 of the data, but samples 1 and n do not. For gradients 1 through n, we know what elements in memory each talk to (both in terms of reads and writes) for any loss parametrized by an inner product. We can for the sake of the proof consider the conflict graph of our n gradients, where an edge is drawn if two gradients affect the same memory element. Intuitively, the chance that concurrent samples interfere increases as the number of samples seen at a given time increases. Furthermore, the chance that two current samples interfere increases as their sparsity decreases. Equivalently, we can say that both an increased number of concurrent samples or decreased sparsity will increase the density of the conflict graph. Both of these concepts will later help choose the value of the parameter τ.

In terms of the bound we wish to compute, we can formalize the notion of two samples interfering as follows. If two samples talk to different variables, $\langle \nabla f_{s_i}(x_i), \nabla f_{s_j}(x_j) \rangle = 0$. If two samples conflict, $\langle \nabla f_{s_i}(x_i), \nabla f_{s_j}(x_j) \rangle \neq 0$. Since we do not know the signs of these inner products, we can immediate use the following bound:

$$\gamma^2 \mathbb{E} \left\langle \sum_{i=k-\tau, i\neq k}^{k+\tau} S_i^k \nabla f_{s_i}(\hat{x}_i), \nabla f_{s_k}(\hat{x}_k) \right\rangle \leq \gamma^2 \mathbb{E} \left| \left\langle \sum_{i=k-\tau, i\neq k}^{k+\tau} S_i^k \nabla f_{s_i}(\hat{x}_i), \nabla f_{s_k}(\hat{x}_k) \right\rangle \right|$$

Next, applying the Cauchy-Schwarz inequality:

$$\gamma^2 \mathbb{E} \left| \left\langle \sum_{i=k-\tau, i\neq k}^{k+\tau} S_i^k \nabla f_{s_i}(\hat{x}_i), \nabla f_{s_k}(\hat{x}_k) \right\rangle \right| \leq \gamma^2 \mathbb{E} \left\{ \sum_{i=k-\tau, i\neq k}^{k+\tau} \|\nabla f_{s_i}(\hat{x}_i)\| \cdot \|\nabla f_{s_k}(\hat{x}_k)\| \mathbf{1}_{s_i \cap s_k = 0} \right\}$$

Where $\mathbf{1}_{s_i \cap s_k = 0}$ is the indicator function of samples i and k overlapping. Note that this expectation only needs to be considered within a τ window of sample k since we make the explicit assumption that at most τ samples are processed while a core processes sample k. Next, we will use the algebraic identity that $a \cdot b \leq \frac{a^2+b^2}{2}$ to get the following

$$\gamma^2 \mathbb{E} \left\{ \sum_{i=k-\tau, i\neq k}^{k+\tau} \|\nabla f_{s_i}(\hat{x}_i)\| \cdot \|\nabla f_{s_k}(\hat{x}_k)\| \mathbf{1}_{s_i \cap s_k = 0} \right\} \leq \gamma^2 \mathbb{E} \left\{ \sum_{i=k-\tau, i\neq k}^{k+\tau} \frac{1}{2} \left(\|\nabla f_{s_i}(\hat{x}_i)\|^2 + \|\nabla f_{s_k}(\hat{x}_k)\|^2 \right) \mathbf{1}_{s_i \cap s_k = 0} \right\}$$

Finally, we employ our assumed bound on the gradient of any sample: $\|\nabla f_s(x)\|^2 \leq M^2$.

$$\gamma^2\mathbb{E}\Bigg\{\sum_{i=k-\tau,i\neq k}^{k+\tau}\frac{1}{2}\Big(\|\nabla f_{s_i}(\hat{x}_i)\|^2+\|\nabla f_{s_k}(\hat{x}_k)\|^2\Big)\mathbf{1}_{s_i\cap s_k=0}\Bigg\}\leq\gamma^2\mathbb{E}\Bigg\{\sum_{i=k-\tau,i\neq k}^{k+\tau}M^2\mathbf{1}_{s_i\cap s_k=0}\Bigg\}$$

$$\leq\gamma^2 2\tau M^2\mathbb{E}\big\{\mathbf{1}_{s_i\cap s_k=0}\big\}$$

Note that $\mathbf{1}_{s_i\cap s_k}=0$ is simply the indicator function of an overlap, so $\mathbb{E}\big\{\mathbf{1}_{s_i\cap s_k=0}\big\}$ is the probability of two samples overlapping. Here, we can consider our conflict graph and apply a result from graph theory to say that:

$$\mathbb{E}\big\{\mathbf{1}_{s_i\cap s_k=0}\big\}=\mathbb{P}\big(s_i\cap s_k\neq 0\big)=\frac{\Delta_{\text{average}}}{n}$$

where Δ_{average} is equal to the average degree of the conflict graph.

Applying this,

$$\gamma^2 2\tau M^2\mathbb{E}\big\{\mathbf{1}_{s_i\cap s_k=0}\big\}=\gamma^2 2\tau M^2\mathbb{P}\big(s_i\cap s_k\neq 0\big)=\gamma^2 2\tau M^2\frac{\Delta_{\text{average}}}{n}$$

Then, summarizing,

$$\gamma\mathbb{E}\{<x_k-\hat{x}_k,\nabla f_{s_k}(\hat{x}_k)>\}=\gamma^2\mathbb{E}\Bigg\langle\sum_{i=k-\tau,i\neq k}^{k+\tau}S_i^k\nabla f_{s_i}(\hat{x}_i),\nabla f_{s_k}(\hat{x}_k)\Bigg\rangle\leq\gamma^2 2\tau M^2\frac{\Delta_{\text{average}}}{n}$$

. To ensure that this entire term is below the desired γ^2M^2, it is necessary to select τ such that $\tau\leq\frac{n}{2\Delta_{\text{average}}}$. If we do so, then we can ensure that our parallel algorithm enjoys the same linear convergence rates as its serial counterpart in the worst case. Note that this is not merely an affect of the proof, but informs our algorithmic design in practice.

16.6 Summary of the Proof and Main Result

Using a parallel SGD algorithm in the manner described above, we can recover the following expected error bounds:

$$\mathbb{E}\big\{\|x_{k+1}-x^*\|^2\big\}\leq(1-\gamma m)\mathbb{E}\big\{\|x_k-x^*\|^2\big\}+\gamma^2\mathbb{E}\big\{\|\nabla f_{s_k}(\hat{x}_k)\|^2\big\}$$

$$+2\gamma m\mathbb{E}\big\{\|x_k-\hat{x}_k\|^2\big\}+2\gamma\mathbb{E}\{<x_k-\hat{x}_k,\nabla f_{s_k}(\hat{x}_k)>\}$$

. These last two terms in the sum can be bounded by careful parameter selection to be on the same order as the standard noise term found in bounds for SGD, $\mathbb{E}\big\{\|\nabla f_{s_k}(\hat{x}_k)\|^2\big\}$. Doing so allows us to apply results from SGD's convergence analysis to this setting and instead treat the parallel version as a noisy version of the serial algorithm. Stated formally,

Theorem 16.1. *If the number of samples that overlap with a single sample during the execution of* HOGWILD! *is bounded as*

$$\tau = \mathcal{O}\left(\min\left\{ \frac{n}{\Delta_{\text{average}}}, \frac{M^2}{\epsilon m^2} \right\} \right)$$

then HOGWILD! *with step size* $\gamma = \frac{\epsilon m}{2M^2}$ *achieves an accuracy of* $\mathbb{E}\|x_k - x^*\|^2 \leq \epsilon$ *after*

$$T \geq \mathcal{O}(1) \frac{M^2 \log \frac{a_0}{\epsilon}}{\epsilon m^2}$$

iterations.

16.7 Examples of Sparse Problems

16.7.1 Sparse Support Vector Machines

Support Vector Machines (SVMs) are useful for classification and other tasks in machine learning. In certain classification tasks, we may have sparse feature vectors. Additionally, Sparse SVMs are useful for dimensionality reduction and feature selection tasks. In all of these tasks, we aim to solve the following minimization:

$$\min_x \sum_{\alpha \in E} \max(1 - y_\alpha x_\alpha^T z_\alpha, 0) + \lambda \|x\|^2$$

If the vectors z_α are sparse, then HOGWILD! is naturally amenable to solving this optimization.

16.7.2 Graph Cuts

Another common example of an optimization where we can expect sparse reads and writes are graph cut problems. These are common in applications such as Topic modeling, entity resolution and image segmentation. For example in Figure 16.4 the image that we want to detect which part of the image contains the fish and which is background. We can represent this image as a graph where pixels are nodes and each contains some (possibly unknown) label indicating whether or not that pixel contains part of the fish. naturally, we expect that the cut of this graph is sparse, i.e., there is a distinct boundary separating the fish from the rest of the image, but the majority of pixels are away from the boundary. This is overlayed in white on top of the image.

16.8 Open Problems

In this analysis, we have fundamentally relied on sparsity and convexity to prove speedups using parallel algorithms. One immediate open problem is how to extend parallelization

Figure 16.4. A figure of a fish in which we want to detect which part of the image contains the fish and which is background. This problem can be described with graph cuts

frameworks, such as HOGWILD! to dense settings. Related to this is the idea of featurizing dense machine learning problems so that they are sparse. This effectively approximates a dense problem as a sparse one. This motivates the open problem of quantifying the fundamental tradeoff between sparsity and learning quality when using parallel algorithms.

Another open problem related to parallel descent methods is quantifying the speedup in terms of time. What we have shown so far is that

$$\text{Worst case speedup} = \frac{\text{bound on \#iter of serial SGD to accuracy } \epsilon}{\text{bound on \#iter of parallel SGD to accuracy } \epsilon}$$

However, we would like to know that

$$\text{speedup} = \frac{\text{time of serial SGD to accuracy } \epsilon}{\text{time of parallel SGD to accuracy } \epsilon}$$

In addition to relaxing sparsity constraints, it is also desirable to relax convexity to prove convergence for non-convex problem.

Finally, this analysis is inherently tied to Shared Memory Systems, where we can assume that the main bottleneck is computation, rather than communication. However, as we scale up and want to consider larger systems, it is necessary to adapt our analysis to consider communication bottlenecks in NUMA and other systems, were the cost of memory access is not uniform. This problem further extends to considering distributed systems, and motivates the open problem of determining the ideal ML paradigm for distributed computing.

Bibliography

[1] Trishul Chilimbi et al. "Project adam: Building an efficient and scalable deep learning training system". In: *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*. 2014, pp. 571–582.

[2] Jeffrey Dean et al. "Large scale distributed deep networks". In: *Advances in neural information processing systems*. 2012, pp. 1223–1231.

[3] Horia Mania et al. "Perturbed iterate analysis for asynchronous stochastic optimization". In: *arXiv preprint arXiv:1507.06970* (2015).

[4] Xinghao Pan et al. "CYCLADES: Conflict-free Asynchronous Machine Learning". In: *arXiv preprint arXiv:1605.09721* (2016).

[5] Benjamin Recht et al. "Hogwild: A lock-free approach to parallelizing stochastic gradient descent". In: *Advances in Neural Information Processing Systems*. 2011, pp. 693–701.