Potential Problem for Problem Set 3

October 20, 2021

1 Problem

We will discover distributed SGD in this question.

Ten years ago, the ML world was all about speed. There were no standard frames like PyTorch and TensorFlow. At that time, distributed computing algorithms like MapReduce was prevalent. Many system folks believed parallel programming would be the answer for ML.

Intuitively, with p processors in parallel, computation should be p times faster. But experimentally, people only found that running time instead increased with p. Indeed, if the processors didn't need to coordinate or communicate, the computation would be p times faster. However, the SGD jobs are highly dependent and sequential. For example, computing the parameter x_{k+1} of the $(k+1)$-th learning iteration requires the previous x_k. In order to avoid overwriting on x_{k+1}, we need to run the locking protocol which will block all the other processors from updating x_{k+1}.

Therefore, distribution frames should never be used for SGD as we talked above. Then, a person deleted all locking protocol codes but still got the right result! Now each processor is lock-free and only needs to run the following code:
Assume we have N sample data $\mathcal{S} = \{s_1, ..., s_N\}$. Let x_k be the parameter computed in the k-th learning iteration, and we sampled s_k from $\mathcal{S}$. Let the gradient on s_k be ∇f_{s_k}, and γ as the learning rate.

- Sample k from $1, 2, ..., N$;
- Read x_k from shared memory, and evaluate gradient ∇f_{s_k} of sample s_k;
- Use ∇f_{s_k} to update x_{k+1} in the shared memory with learning rate γ, as $x_{k+1} = x_k - \gamma \nabla f_{s_k}$.

Clearly, x_{k+1} might overlap with other processors. But with the following analysis, we will guarantee the accuracy of lock-free distributed SGD, with a near linear speed improvement.

In these questions, we assume $f_{s_k}(x)$ is m-strongly convex $x \mapsto f(x) - \frac{m}{2}\|x\|_2^2$ with $\|\nabla f_{s_k}(x)\|^2 \le M^2$. Unlike lecture notes, we are trying to guarantee $\mathbb{E}\|\hat{x} - x^*\|^2 \le \epsilon$, which is just a slightly different metric for analysis convenience.

(a) (Warm up questions)

- Compute how many iterations do we need for converge under the condition $\mathbb{E}\|\hat{x} - x^*\|^2 \le \epsilon$?
- For classical distributed SGD with locking, a fact is one processor finishes writing the update need to notify all the other $p-1$ processors. And p processors will need $(p/2)^2$ to communicate. Explain why the time cost of distributed SGD with locking is more than the non-parallel SGD?

(b) Lock-free parallel SGD is one kind of asynchronous algorithm. Since we learned SGD algorithm originally have some robustness of noise. We will treat overlap writing as a kind of noise. Let's define s_k to be the k-th sampled data point. The principle is that cores do not read the "actual" iterate x_k but the "noisy" iterate $\hat{x}_k := x_k + noise$. After T processed samples, the shared memory contains:

$$\hat{x}_T := x_0 - \gamma \nabla f_{s_0}(\hat{x}_0) - ... - \gamma \nabla f_{s_{T-1}}(\hat{x}_{T-1}).$$

We want to prove the accuracy $\mathbb{E}\|x_T - x^*\|^2 \leq \epsilon$ to claim convergence. We will do a similar analysis as the lecture notes.

- Prove the following inequality:

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

- Let $2\gamma m\mathbb{E}\{\|x_k - \hat{x}_k\|\}$ be L_1, and $2\gamma\mathbb{E}\{\langle x_k - \hat{x}_k, \nabla f_{s_k}(\hat{x}_k)\rangle\}$ be L_2. Prove that if L_1 and L_2 are both $\mathcal{O}(\gamma^2 M^2)$, noisy SGD gets same convergence rates as the SGD up to multiplicative constants.

(c) Part (2) claimed SGD is robust to small perturbations. In this question we are going to prove that when our dataset is sparse we have $L_1, L_2 \leq \mathcal{O}(\gamma^2 M^2)$. The noise of lock-free parallel mainly comes from 2 places: First, if one processor computes ∇f_{s_i} before s_k is sampled: its gradient contribution is recorded in shared memory, while a thread starts working on s_k. So the gradient from s_k will arrive too late. Second, if s_i overlaps in time with s_k (i.e. the two samples are concurrently processed): its gradient contribution might only be partially recorded in shared memory, when a thread starts working on s_k.

While any processor is processing a sample, assume no more than τ samples processed by other processors. For each sample s_k, any difference between $\hat{x}_k$ and x_k is caused only by samples that "overlap" with s_k in one of the two above mentioned ways. Therefore, if x_i is sampled before s_k, it might overlap with s_k if and only if $i \geq k - \tau$; if s_i is sampled after s_k, it might overlap with s_k if and only if $i \leq k + \tau$. Shown in the following graph.

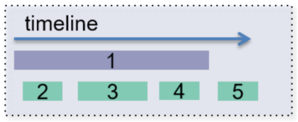

Figure 1: If $\tau = 3$, while 1 is being processed no more than 3 updates occur

Let the probability of s_i overlap with s_k is fixed as $Pr\{s_i \cap s_k \neq 0\} = c$.

- Express $\hat{x}_k - x_k$ in terms of $\nabla f_{s_i}(\hat{x}_i)$.
- Prove $L_1, L_2 \leq \mathcal{O}(\gamma^2 M^2)$, when $\tau \leq \frac{1}{2c}$.

(d) Conclude the result with learning rate $\gamma = \frac{\epsilon m}{2M^2}$ after $T \geq \mathcal{O}(\frac{M^2 \log(\|x_0 - x^*\|^2/\epsilon)}{\epsilon m^2})$ iterations, we have accuracy $\mathbb{E}\|x_T - x^*\|^2 \leq \epsilon$.