Parallel Machine Learning: Lecture Notes

William Cohen

March 12, 2015

1 Notation

$\mathcal{T}$	a dataset
$\mathbf{x}_t, y_t$	the t-th example in a dataset
S	the number of "shards" of data
$\mathcal{T}_i$	the i-th shard
$\mathbf{w}$	a linear classifier/weight vector
$\mathbf{w}$	a weight vector for an on-line learner at time t
$\ell_t(\mathbf{w}, \mathbf{x}_t, y_t)$	the loss incurred by $\mathbf{w}$ on t-th example
$\mathbf{g}_t$	gradient of loss wrt $\mathbf{w}$ at $\mathbf{x}_t$
τ	a "delay" (# time steps out of date)
L	bound on subgradient of loss

2 Adapting Streaming Learning Methods to Map-Reduce

In class we covered two closely-related streaming learning algorithms in detail: logistic regression using stochastic gradient descent (SGD), and the averaged perceptron algorithm. We also discussed in detail one paper discussing parallelizing the perceptron algorithm.[1] That paper describes two parallel algorithms.

- The *parameter mixing* (PM) version of the perceptron runs the program on S "shards" of data, each a randomly-chosen subset of the instances, and then averages the S different linear classifiers that are learned.

[1] Distributed Training Strategies for the Structured Perceptron, R. McDonald, K. Hall and G. Mann, North American Association for Computational Linguistics (NAACL), 2010.

- The *iterative parameter mixing* (IPM) version of the perceptron does parameter mixing in a loop. At the end of the k-th loop, the "mixed" (averaged) linear classifier $\mathbf{w}^(k)$ is broadcast to each of the S worker processors, and they stream through the data again, using this as the starting point for the on-line perceptron perceptron algorithm.

Formally, McDonald et al. show (1) that PM will fail to converge in the worse case; and that (2) IPM will converge, but in the worst case, might take S times as many mistakes to converge—i.e., might be no faster than a single on-line worker. The second result was presented in detail in class.

Experimentally, however, IPM works fairly well, providing a wall-clock speedup of 25x or so over a serial process. The experiments suggest that parameter mixing is not effective: it seems to work less well than a single iteration. Formally, it's left open whether the worst-case results in the mistake-bound setting really reflect performance on the i.i.d. data one would expect to see in practice.

todo: discuss experiments, and questions they raise. does it beat subsampling? is it faster than a serial method? how scalable is it?

3 Delayed SGD

PM and IPM are prototypical of two common approaches to parallel ML in a map-reduce like setting, which one could instantiate in a number of other settings: in particular, you could use the same approaches for any streaming machine learning method, like as SGD. PM provides maximal parallelism, and requires minimal communication. IPM requires more communication but may provide better performance.

In another well-cited paper[2] the idea of iterative parameter mixing is extended as follows. In IPM, the essential idea is that parallelism is enabled because different worker processes are *allowed to let their weight vectors diverge*: i.e., each process is using a "stale", "out of date" version of the weight vector. Langford et al. consider *delayed stochastic gradient descent* (DSGD), an on-line SGD process where in each step, an old version of the weight vector is used to compute a gradient. Formally, if τ is a delay, $\mathbf{g}_t$ is the usual SGD gradient at time t, and η_t is the learning rate at t, a weight vector will

[2] "Slow learners are fast", Langford, Smola, Zinkevick JMLR 2009

be updated at time t to

$$\mathbf{w}_t = \mathbf{w}_t - \eta_t \mathbf{g}_{t-\tau}$$

i.e., using a gradient that was computed τ steps previously.

Delayed SGD is a formal model, which could be used to analyze an IPM-like version of SGD—here τ would grow as each worker node progressed. Another reasonable parallel algorithm that can be analyzed in this framework is a multi-threaded architecture where $\mathbf{w}$ is shared by many worker threads, each of which repeatedly reads (part of) $\mathbf{w}$, computes a gradient, and then sends an update (which may by now be out-of-date, due to other worker updates) to a process that controls write access to $\mathbf{w}$.

Comments on the theory. What can you say about this model? As noted above, it's close to the IPM setting, which in the mistake-bounded model converges, but might converge as slowly as a serial computation that consumes a fixed fraction $\frac{1}{S}$ of the data in each iteration. The McDonald et al. results clearly depend on the mistake-bound model, and it's not obvious how to extend them to an SGD setting.

To summarize the "Slow learners are fast" theoretical results, the notion of a mistake bound is extended to "regret". Let $\mathbf{w}_t$ be the t-th weight vector, and define $\ell_t(\mathbf{w}_t, \mathbf{x}_t, y_t)$ to be the loss incurred by $\mathbf{w}_t$ on the t-the example. Let $\mathbf{w}^*$ be the weight vector that minimizes loss over the entire dataset $\mathcal{T}$. We assume that $\mathcal{T}$ is ordered, and now the *regret* for the dataset $\mathcal{T}$ is defined as

$$R[\mathcal{T}] \equiv \sum_{t=1}^{T} \ell_t(\mathbf{w}_t, \mathbf{x}_t, y_t) - \ell_t(\mathbf{w}^*, \mathbf{x}_t, y_t)$$

If ℓ_t is 0-1 classification error and $\mathcal{T}$ is linearly seperable, then regret reduces to the number of mistakes made: however, it is much more general. Roughly speaking, regret is how well the on-line process performs relative to a batch process in which all the examples are available, and like a mistake bound, a bound on regret is closely related to generalization performance in an i.i.d. setting.

To bound regret in a delayed SGD setting, you need to make some assumptions. We will assume

1. The loss is convex. Convexity means that we can bound

$$R[\mathcal{T}] \equiv \sum_{t=1}^{T} \ell_t(\mathbf{w}_t, \mathbf{x}_t, y_t) - \ell_t(\mathbf{w}^*, \mathbf{x}_t, y_t) \leq \sum_{t=1}^{T} \langle \mathbf{g}_t, \mathbf{w}_t - \mathbf{w}^* \rangle$$

   where $\langle \mathbf{u}, \mathbf{v} \rangle$ is the inner product.

2. The gradient (more technically, the subgradients) of the loss are bounded by a constant L, i.e., for all t, $\mathbf{g}_t \leq L$ for some $L > 0$.

todo: mistake bound goes to "regret", and you need to assume something about how far off a "stale" update can be. which needs assumptions about smoothness of loss - convex and Lipschitz continuous, where loss gradient is bounded by L – and a distance function between parameters – bregman divergence, which generalizes an inner product – so we can bound in instantaneous loss.

bounds range from $O(\tau \log T)$ to $O(\tau^2 + \log T)$ for strongly convex losses and smooth gradients.
