Parallelization in the "Big Data" Regime: Model Parallelization?

Sham M. Kakade

Machine Learning for Big Data
CSE547/STAT548

University of Washington

Announcements...

- HW 3 due Mon (NIPS extension)
- Projects: the term is approaching the end....

Today:

- Review: mini-batching
- Overview:
  1. Averaging
  2. Hogwild/asynchrony
  3. Model parallelization/coordinate ascent
  4. Deep learning

Review: One machine or a cluster?

- One machine:
  - Certain operations are much faster to do when specified without “for loops”: matrix multiplications, convolutions, Fourier transforms,
  - GPUs!!!
  - Shared memory/communication is fast! Try to take advantage of fast “simultaneous” operations.
- Cluster:
  - Why? One machine can only do so much.
  - Try to (truly) breakup computations to be done.
  - Drawbacks: Communication is costly!
  - Simple method: run multiple jobs (or models) with different parameters.

Review: Data Parallelization vs. Model parallelization

- Data parallelization:
  Breakup data into smaller chunks to process.
  - Mini-batching, batch gradient descent
  - Averaging
- Model parallelization:
  Breakup up your model.
  - Try to update parts of model in a distributed manner.
  - Coordinate ascent methods
  - Update layers of a neural net on different machines.
- Other issues:
  Asynchrony

Review: Mini-batching and the "critical" batch size b

- Let $\tilde{b}$ be the critical b in which η_b^* is $\eta_\infty^*/2$.
- Informal Theorem: (square loss case) Up until $\tilde{b}$, you will linear improvements in you serial complexity (with minor changes in the total work). (Asymptotically, you can mini-batch to any extent.)
- Does this hold more generally?
- Theorem: Suppose $\|x\|^2 \leq L$ (almost surely) and λ_{max} is the maximal eigenvalue of $\mathbb{E}[xx^\top]$. Then:

$$\frac{\mathbb{E}[\|x\|^2]}{\lambda_{\text{max}}} \leq \tilde{b} \leq \frac{L}{\lambda_{\text{max}}}$$

(for not very kurtotic distributions, $L \approx \mathbb{E}[\|x\|^2]$).

Topic 1: Averaging & Data-Parallelization

- With “mini-batching”, our presentation suggests you might as well just use a single machine with mini-batching.
- What can we do with multiple machines?
- In the convex case:
  - Breakup your data onto M machines
    (Must still have “enough” data on each machine.)

$$\mathcal{D} = \mathcal{D}_1 \cup \mathcal{D}_2 \cup \ldots \cup \mathcal{D}_M$$

  - Run (mini-batch) SGD separately on each machine separately.
  - Communicate each machines anser to a central “parameter server”, and (by convexity) average the final answer from each machine.
  - Then repeat.

Averaging: How good is it?

- Question: What if there isn't enough data on each machine?
- How much data do we need on each machine?
  Roughly, we need κ data points per machine.
- Theorem: With "enough" data on each machine, then one can just run one pass of SGD separately on each machine and then average their answers. This will be optimal statistically (e.g. in terms of generalization).

Topic 2: Hogwild and Asynchronous Updating

- mini-batch SGD: using batch size b:

$$w_{t+1} \leftarrow w_t - \eta_b \left(\frac{1}{b} \sum_{j=1}^{b} \widehat{\nabla \ell_j(w_t)} \right)$$

  where η_b is our learning rate.
- Suppose we parallelize this on b machines:
  - Each machine computes $\frac{\eta_b}{b} \widehat{\nabla \ell_j(w_t)}$.
    - They read w_t from a "parameter server".
    - They wait until w_t is updated before reading again.
  - To compute w_{t+1}, we need to do b additions of the form:

$$w \leftarrow w - \frac{\eta_b}{b} \widehat{\nabla \ell_j(w_t)}$$

  - Each machine "locks" the parameter server when updating.
- Problem: these are serial and "locks" can be slow.

Hogwild and Asynchronous Updating

- Hogwild:
  - Ignore the locks.
  - Each machine reads the current "w" when it wants.
  - Each machine updates on the parameter server, ignoring the locks.
- Informal Theorem: Suppose the updates, $\frac{\eta_b}{b}\widehat{\nabla \ell_j(w_t)}$, computed by each machine are sparse (sufficiently sparse and without much conflicts). Then some guarantees on accuracy.
- Does it work more generally?

Topic 3: Model Parallelization

- Try to update model parameters simultaneously, on different machines.
- Recall coordinate ascent on a loss function $L(w)$:
  - start with some w.
  - choose coordinate i randomly, and update:

$$\begin{aligned} \Delta_i &= \operatorname{argmin}_\Delta F(w_1, \ldots w_{i-1}, w_i + \Delta, \ldots, w_d) \\ w_i &\leftarrow w_i + \Delta_i \end{aligned}$$

  - the complexity depends on the argmin.
  - return w
- Idea: What if we try to update many coordinate in parallel?

Example: L1/Shotgun

- For the Lasso, the argmin can be done efficiently.
- Compute the Δ_i's for b coordinates in parallel.
- How do we update each w_i?
  We must use some stepsize η

$$w_i \leftarrow w_i + \eta \Delta_i$$

  (where $\eta \geq 1/b$).
- Much like "mini-batching", this is helpful (up to a point).

Model Parallelization (in the dual)

- instead of optimizing $L(w_1, \ldots w_d)$, let's optimize the dual function $G(\alpha_1, \ldots \alpha_{i-1}, z, \ldots, \alpha_n)$.
- now coordinates α_i are associated with datapoints x_i.
- So model parallelization/coordinate ascent is essentially data parallelization.
  We are updating parameters associated with datapoints.
- How much does this help?
  Again, much like mini-batching.
- Practice?