Optimization in the “Big Data” Regime

Sham M. Kakade

Machine Learning for Big Data
CSE547/STAT548

University of Washington

Announcements...

- HW2 due Mon.
- Work on your project milestones
  - read/related work summary
  - some empirical work

Today:

- Review: discuss classical optimization
- New: How do we optimize in the "big data" regime, with large sample sizes and large dimension?
  Bridge classical to modern optimization.

Machine Learning and the Big Data Regime...

goal: find a d-dim parameter vector which minimizes the loss on n training examples.

- have n training examples $(x_1, y_1), \ldots (x_n, y_n)$
- have parametric a classifier $h_\theta(x, w)$, where w is a d dimensional vector.

$$\min_w L(w) \text{ where } L(w) = \sum_i \text{loss}(h(x_i, w), y_i)$$

- "Big Data Regime": How do you optimize this when n and d are large? memory? parallelization?

Can we obtain linear time algorithms to find an ϵ-accurate solution?

i.e. find $\hat{w}$ so that

$$L(\hat{w}) - \min_w L(w) \leq \epsilon$$

Plan:

- Goal: algorithms to get fixed target accuracy ϵ.
- Review: classical optimization viewpoints
- A modern view: can be bridge classical optimization to modern problems?
  - Dual Coordinate Descent Methods
  - Stochastic Variance Reduced Gradient method (SVRG)

Abstraction: Least Squares

$$\min_{w} L(w) \text{ where } L(w) = \sum_{i=1}^{n} (w \cdot x_i - y_i)^2 + \lambda \|w\|^2$$

How much computation time is required to to get ϵ accuracy?

- *n* points, *d* dimensions.
- "Big Data Regime": How do you optimize this when *n* and *d* are large?
- More general case: Optimize sums of convex (or non-convex functions?
    - some guarantees will still hold

Aside: think of *x* as a large feature representation.

Review: Direct Solution

$$\min_{w} L(w) \text{ where } L(w) = \sum_{i=1}^{n} (w \cdot x_i - y_i)^2 + \lambda \|w\|^2$$

- solution:

$$\vec{w} = (X^\top X + \lambda I)^{-1} X^\top Y$$

  where X be the $n \times d$ matrix whose rows are x_i, and Y is an n-dim vector.
- numerical solution: the “backslash” implementation.
- time complexity: $O(nd^2)$ and memory $O(d^2)$

Not feasible due to both time and memory.

Review: Gradient Descent (and Conjugate GD)

GD: $w \leftarrow w - \eta \nabla L(w)$, $\eta = \frac{1}{\lambda_{max}}$

$$\min_w L(w) \text{ where } L(w) = \sum_{i=1}^{n} (w \cdot x_i - y_i)^2 + \lambda \|w\|^2$$

- n points, d dimensions,
- $\lambda_{\max}, \lambda_{\min}$ are max and min eigs. of "design matrix" $\frac{1}{n} \sum_i x_i x_i^\top$ $= X^T X$
- # iterations and computation time to get ϵ accuracy:
  - Gradient Descent (GD):

$$\frac{\lambda_{\max}}{\lambda_{\min}} \log 1/\epsilon, \quad \frac{\lambda_{\max}}{\lambda_{\min}} nd \log 1/\epsilon$$

$X \in \mathbb{R}^{n \times d}$

  - Conjugate Gradient Descent:

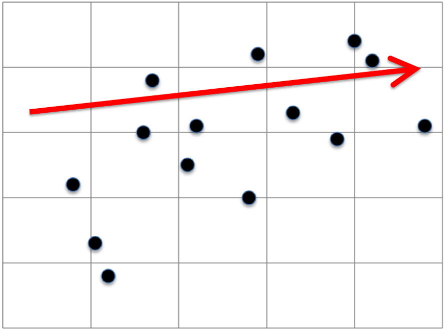
$$\sqrt{\frac{\lambda_{\max}}{\lambda_{\min}}} \log 1/\epsilon, \quad \sqrt{\frac{\lambda_{\max}}{\lambda_{\min}}} nd \log 1/\epsilon$$

$= \left[x_1 \middle| x_2 \middle| \cdots x_n \right]$

- memory: $O(d)$

Better runtime and memory, but still costly.

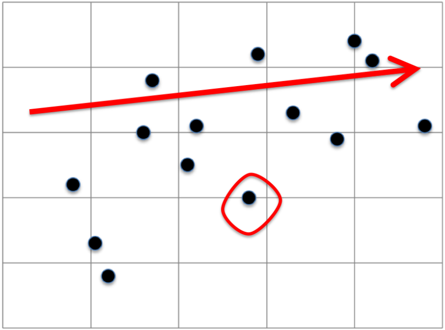
Review: Stochastic Gradient Descent (SGD)



- SGD update rule: at each time t,

$$\text{sample a point } (x_i, y_i)$$
$$w \leftarrow w - \eta(w \cdot x_i - y_i)x_i$$

Review: Stochastic Gradient Descent (SGD)

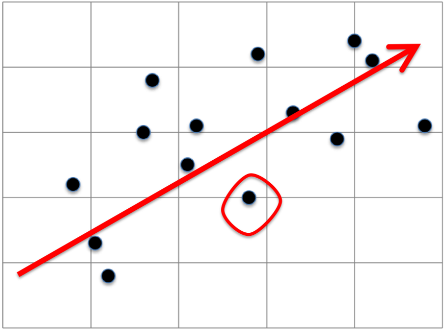


- SGD update rule: at each time t,

$(\lambda = 0)$

$$\text{sample a point } (x_i, y_i)$$
$$w \leftarrow w - \eta(w \cdot x_i - y_i)x_i$$

Review: Stochastic Gradient Descent (SGD)



at opt.

$d_i = y_i - w^* x_i$

- SGD update rule: at each time t,

$$\text{sample a point } (x_i, y_i)$$
$$w \leftarrow w - \eta(w \cdot x_i - y_i)x_i$$

- Problem: even if $w = w_*$, the update changes w.
  Rate: convergence rate is $O(1/\epsilon)$, with decaying η
  simple algorithm, light on memory, but poor convergence rate

Review: Stochastic Gradient Descent

- $\lambda_{\min}$ is the min eig. of $\frac{1}{n}\sum_i x_i x_i^\top$
- Suppose gradients are bounded by B.
- To get ϵ accuracy:
  - # iterations to get ϵ-accuracy:

$$\frac{B^2}{\lambda_{\min}\epsilon}$$

  - Computation time to get ϵ-accuracy:

$$\frac{dB^2}{\lambda_{\min}\epsilon}$$

$O\left(\frac{1}{\varepsilon}\right)$.

$\underline{\underline{\text{not}}}$

$O\left(\log \frac{1}{\varepsilon}\right)$

Regression in the big data regime?

$$\min_{w} L(w)$$

How much computation time is required to to get ϵ accuracy?

- "Big Data Regime": How do you optimize this when n and d are large?
  - Can we 'fix' the instabilities of SGD?
- Let's look at (regularized) linear regression.
  - Convex optimization: All results can be generalized to smooth+strongly convex loss functions.
  -
  - Non-convex optimization: some ideas generalize.

Duality (without Duality)

$X \in \mathbb{R}^{n\times d} = [x_1 | \cdots | x_n]^T$ $Y \in \mathbb{R}^n$

$$\begin{aligned} w^* &= (X^\top X+\lambda I)^{-1}X^\top Y \\ &= X^\top \underbrace{(XX^\top+\lambda I)^{-1} Y} \\ &:= \frac{1}{\lambda}X^\top \alpha \end{aligned}$$

$XX^\top \in \mathbb{R}^{n\times n}$

where $\alpha = (I + XX^\top/\lambda)^{-1} Y$.

def.

not

invertible

- idea: let's compute the n-dim vector α.
- let's do this with coordinate ascent

$\alpha = A^{-1} b$

$G(\alpha) = (A\alpha - b)^2$ or

$\alpha A \alpha - b\alpha$

SDCA: stochastic dual coordinate ascent

$$G(\alpha_1, \alpha_2, \ldots \alpha_n) \overset{\text{lof}}{:=} \frac{1}{2}\alpha^\top (I + XX^\top/\lambda)\alpha - Y^\top \alpha$$

- the minimizer of $G(\alpha)$ is

$$\alpha = (I + XX^\top/\lambda)^{-1} Y$$

- SDCA:
  - start with $\alpha = 0$.
  - choose coordinate i randomly, and update:

$$\alpha_i = \operatorname{argmin}_z G(\alpha_1, \ldots \alpha_{i-1}, z, \ldots, \alpha_n)$$

  - easy to do as we touch just one datapoint.
  - return $w = \frac{1}{\lambda} X^\top \alpha$.

SDCA: the algorithm

$$G(\alpha_1, \alpha_2, \ldots \alpha_n) = \frac{1}{2}\alpha^\top (I + XX^\top/\lambda)\alpha - Y^\top \alpha$$

- start with $\alpha = 0$, $w = \frac{1}{\lambda}X^\top \alpha$.
  1. choose coordinate i randomly, and compute difference:

     $$\Delta\alpha_i = \frac{(y_i - w \cdot x_i) - \alpha_i}{1 + \|x_i\|^2/\lambda}$$

  2. update:

     $$\alpha_i \leftarrow \alpha_i + \Delta\alpha_i, \quad w \leftarrow w + \frac{1}{\lambda}x_i \cdot \Delta\alpha_i$$

- return $w = \frac{1}{\lambda}X^\top \alpha$.

at w^* what is α_i ??

$$\boxed{\alpha_i = y_i - w^* x_i}$$

Guarantees: speedups for the big data regime

- n points, d dimensions, λ_{av} average eigenvalue
- Computation time to get ϵ accuracy gradient descent: (Shalev-Shwartz & Zhang '12)
  - GD vs SDCA:

$$\frac{\lambda_{\max}}{\lambda_{\min}} n\, d \log 1/\epsilon \to \left(n + d \frac{\lambda_{\text{av}}}{\lambda_{\min}}\right) d \log 1/\epsilon$$

  - conjugate GD vs acceleration+SDCA.
    One can accelerate SDCA as well. (Frosting, Ge, K., Sidford, 2015))

iters.

$$\left(n + d \frac{\Sigma}{\lambda_{min}}\right) \log \frac{1}{\varepsilon}$$

Comparisons to SGD

SDCA / SGD

- both algorithms touch one data point at a time, with same computational cost per iteration.
- SDCA has “learning rate” which adaptive to the data point.
- SGD has convergence rate of $1/\epsilon$ and SDCA has $\log 1/\epsilon$ convergence rate.
- memory: SDCA: $O(n+d)$, SGD: $O(d)$
- SDCA: can touch points in *any* order.

SDCA advantages/disadvantages

- What about more general convex problems? e.g.

$$\min_w L(w) \text{ where } L(w) = \sum_i \mathrm{loss}(h(x_i, w), y_i)$$

  - the basic idea (formalized with duality) is pretty general for convex $\mathrm{loss}(\cdot)$.
  - works very well in practice.
- memory: SDCA needs $O(n + d)$ memory, while SGD is only $O(d)$.
- What about an algorithm for non-convex problems?
  - SDCA seems heavily tied to the convex case.
  - would an algo that is highly accurate in the convex case and sensible in the non-convex case.

(another idea) Stochastic Variance Reduced Gradient (SVRG)

1. exact gradient computation: at stage s, using $\widetilde{w}_s$, compute:

$$\nabla L(\widetilde{w}_s) = \frac{1}{n}\sum_{i=1}^{n} \nabla \text{loss}(\widetilde{w}_s, (x_i, y_i))$$

2. corrected SGD: initialize $w \leftarrow \widetilde{w}_s$. for m steps,

$$\text{sample a point } (x, y)$$
$$w \leftarrow w - \eta \left(\nabla \text{loss}(w, (x, y)) - \nabla \text{loss}(\widetilde{w}_s, (x, y)) + \nabla L(\widetilde{w}_s) \right)$$

3. update and repeat: $\widetilde{w}_{s+1} \leftarrow w$.

(another idea) Stochastic Variance Reduced Gradient (SVRG)

1. exact gradient computation: at stage s, using $\widetilde{w}_s$, compute:

$$\nabla L(\widetilde{w}_s) = \frac{1}{n}\sum_{i=1}^{n} \nabla \mathrm{loss}(\widetilde{w}_s, (x_i, y_i))$$

2. corrected SGD: initialize $w \leftarrow \widetilde{w}_s$. for m steps,

$$\text{sample a point } (x, y)$$
$$w \leftarrow w - \eta \left(\nabla \mathrm{loss}(w, (x, y)) - \nabla \mathrm{loss}(\widetilde{w}_s, (x, y)) + \nabla L(\widetilde{w}_s) \right)$$

3. update and repeat: $\widetilde{w}_{s+1} \leftarrow w$.

Two ideas:

- If $\widetilde{w} = w_*$, then no update.
- unbiased updates: blue term is mean 0.

Guarantees of SVRG

- *n* points, *d* dimensions, λ_{av} average eigenvalue
- Computation time to get ϵ accuracy gradient descent: (Johnson & Zhang ’13)
  - GD vs SDCA:

$$\frac{\lambda_{\max}}{\lambda_{\min}} n\, d \log 1/\epsilon \to \left(n + d\frac{\lambda_{\text{av}}}{\lambda_{\min}}\right) d \log 1/\epsilon$$

  - conjugate GD vs ??

$$\sqrt{\frac{\lambda_{\max}}{\lambda_{\min}}}\, n\, d \log 1/\epsilon \to ??$$

  - memory: $O(d)$